ORSAY
Série A
N° d'ordre
175

THESE

PRESENTEE

A L'UNIVERSITE DE PARIS-SUD
CENTRE D'ORSAY

POUR OBTENIR

LE TITRE DE DOCTEUR INGENIEUR

par

Jean-Jacques CHAUVIN

CONCEPTION ET MISE AU POINT D'UN COMPILATEUR BASIC SUR GE 55

Soutenue le 24 novembre 1972, devant la Commission d'Examen

M. DEBRAINE — Président

M. DONADIEU
Mle BESTOUGEFF — Examinateurs

ORSAY
Série A
N° d'ordre

175

THESE

PRESENTEE

A L'UNIVERSITE DE PARIS-SUD
CENTRE D'ORSAY

POUR OBTENIR

LE TITRE DE DOCTEUR INGENIEUR

par

Jean-Jacques CHAUVIN

CONCEPTION ET MISE AU POINT D'UN COMPILATEUR BASIC SUR GE 55

Soutenue le 24 novembre 1972, devant la Commission d'Examen

M. DEBRAINE — Président

M. DONADIEU
Mle BESTOUGEFF — Examinateurs

UNIV.
LEIDEN
BIBL.

Avant de présenter ce travail, je tiens à exprimer ma profonde reconnaissance à Monsieur P. DEBRAINE, Professeur Associé à l'Université de PARIS-SUD, Professeur à l'Institut National des Sciences et Techniques Nucléaires, Adjoint du Directeur du Centre d'Etudes Nucléaires de SACLAY, d'avoir bien voulu m'accepter dans son Service et de me faire l'honneur de présider le jury.

Je tiens à remercier Mademoiselle H. BESTOUGEFF, Maître de Conférences à l'Université de PARIS VII, et Monsieur le Professeur L. DONADIEU, Directeur de l'Institut Universitaire de Technologie d'Orsay, qui malgré leurs nombreuses occupations, ont bien voulu accepter d'être membres du jury.

Je tiens à remercier Messieurs LECLERC et RIVIERE, Ingénieurs à la Compagnie HONEYWELL-BULL pour les renseignements qu'ils ont bien voulu me fournir.

Je tiens à exprimer mes remerciements à Mesdames AIGUESPARSES et COHEN ainsi qu'au personnel du Service de Documentation du CEN-S qui ont permis la réalisation et la publication de ce travail dans de bonnes conditions.

TABLE DES MATIERES

		Page
	AVANT-PROPOS	1
	INTRODUCTION	2
I -	COMPILATEURS	7
	I-1 - Traducteurs	7
	I-2 - Compilateur génératif	9
	I-3 - Compilateur interprétatif	10
	I-4 - Remarques	11
	I-5 - Comparaisons	11
II -	LANGAGES - GRAMMAIRES	13
	II-1 - Généralités	13
	II-2 - Langages	13
	II-3 - Grammaires	16
	II-4 - Classification des grammaires	20
	II-5 - Exemple - Représentation de Backus-Naur	21
	II-6 - Graphe de dérivation - Arbre syntaxique	23
	II-7 - Analyse syntaxique	25
III -	DESCRIPTION DE LA MACHINE	26
	III-1 - Configuration	26
	III-2 - Software	26
	III-3 - Unité centrale	27
	III-4 - Instructions	33
	III-5 - Basic Opérating system/tambour	34
	III-6 - Représentation flottante	35
IV -	ANALYSE SYNTAXIQUE	37
	IV-1 - Rappels	37
	IV-2 - Expressions arithmétiques - Introduction aux méthodes pratiques	37
	IV-3 - Méthodes d'analyse	40
	IV-4 - Analyse descendante	43
	IV-5 - Analyse ascendante	46
V -	ANALYSEUR SYNTAXIQUE DU BASIC	70
	V-1 - Définition de la grammaire BASIC	70
	V-2 - Contraintes supplémentaires de BASIC	85
	V-3 - Graphe d'ordre partiel	86
	V-4 - Caractéristiques de l'analyseur	88
	V-5 - Organigrammes de l'analyseur	95

Page

VI - INTERPRETEUR 135

VI-1 - Analyse programme 135
VI-2 - Traitement des DATA 138
VI-3 - Traitement des variables simples 141
VI-4 - Pile d'interprétation. Pile auxiliaire 148
VI-5 - Traitement des quantités numériques 150
VI-6 - Traitement des variables indicées et des matrices 154
VI-7 - Traitement des fonctions 168
VI-8 - Opérations arithmétiques 185
VI-9 - Prise en charge du programme BASIC 194
VI-10 - Instructions READ et INPUT 196
VI-11 - Instruction RESTORE 203
VI-12 - Ordre LET 206
VI-13 - Ordre GOTO 209
VI-14 - Ordres FOR et NEXT. Itérations 210
VI-15 - Ordre IF ... THEN. Branchement conditionnel 218
VI-16 - Ordres GOSUB et RETURN.
Appels et retours de sous-programmes 220
VI-17 - Ordre DEF. Définition de fonction interne 223
VI-18 - Instructions non exécutables. Ordres DATA et REM 223
VI-19 - Instruction de réservation de tableau. Ordre DIM 223
VI-20 - Instruction de sortie. Ordre PRINT 227
VI-21 - Instruction de fin d'exécution. Ordres STOP et END 232
VI-22 - Instructions matricielles.
Ordres MAT, MATINPUT, MATREAD, MATPRINT 232
VI-23 - Interruptions normales et erreurs 254

VII - PRATIQUE DU BASIC SUR GE 55 257

VII-1 - Introduction du compilateur en machine 257
VII-2 - Analyseur syntaxique 260
VII-3 - Interpréteur 260
VII-4 - Organisation de la mémoire centrale et du tambour
en interprétation 264
VII-5 - Contraintes de l'interpréteur 267
VII-6 - Temps d'exécution 268

VIII - EXEMPLES DE PROGRAMMES BASIC EXECUTES SUR GE 55 272

VIII-1 - Liste des exemples 272
VIII-2 - Tableau de distribution des instructions 273
VIII-3 - Exemples 275

CONCLUSION 298

Page

ANNEXE A - LISTE ENCHAINEE DES MOTS-CLES 301

A-1 - Principes 301
A-2 - Création 304
A-3 - Utilisation 306

ANNEXE B - PROGRAMME DE TRI EN TAMBOUR 316

B-1 - Nécessité d'un tri en tambour 316
B-2 - Environnement de la séquence de tri 317
B-3 - Principe de l'enregistrement des blocs en tambour 317
B-4 - Modification de la dernière ligne enregistrée 325
B-5 - Elimination d'une ligne antérieure 325
B-6 - Remplacement d'une ligne antérieure 326
B-7 - Insertion d'une nouvelle ligne 326
B-8 - Elimination d'une ligne inexistante 326
B-9 - Remarques 328
B-10- Commandes LIST et PUNCH 328

ANNEXE C - LISTE DES ERREURS DE SYNTAXE 329

ANNEXE D - EXEMPLES D'ARBRES D'ANALYSE SYNTAXIQUE 335

D-1 - Introduction - Remarques générales 335
D-2 - Exemples 335

ANNEXE E - RECHERCHE DU CARACTERE ENTIER DE L'EXPOSANT POUR A↑X 350

E-1 - Introduction 350
E-2 - Caractère entier 350
E-3 - Parité 352

ANNEXE F - SOUS-PROGRAMME DE CONVERSION EXTERNE EXTFLOCH 355

F-1 - But 355
F-2 - Types de format 355
F-3 - Traitement 356

ANNEXE G - LISTE DES ERREURS D'INTERPRETATION 360

BIBLIOGRAPHIE 363

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41549225_0004

AVANT-PROPOS

L'étude proposée ici concerne la conception et la mise au point d'un compilateur BASIC sur GE 55.

Le BASIC est un langage de programmation de type évolué. Il possède un jeu d'instructions classiques relatives aux boucles, aux fonctions, aux branchements, etc... Par ailleurs, il possède d'une part, un jeu d'instructions permettant d'utiliser le calcul matriciel, et d'autre part, des instructions d'entrée directe de données en cours d'exécution du programme. Ce dernier type d'instructions rend le BASIC conversationnel.

Le dialogue entre un programme et un opérateur en mode conversationnel s'effectue d'habitude au moyen d'un télétype. La machine pour laquelle cette étude est envisagée ne permettant pas une connexion de ce type, il faut ici, simuler le fonctionnement d'un télétype à l'aide, d'une part, d'un clavier alphanumérique d'entrée directe et, d'autre part, d'une imprimante.

Le travail de programmation a été réalisé sur un ordinateur GE 55 de la Société HONEYWELL-BULL installé temporairement au C.E.N. de Saclay.

INTRODUCTION

Un ordinateur est une machine numérique de traitement de l'information. Son fonctionnement de base est assuré par des organes électroniques élémentaires, les plus significatifs étant les portes, les bascules et les mémoires à ferrite, travaillant en code binaire, c'est-à-dire susceptibles de reconnaître deux types de signaux habituellement codés 0 et 1, ces signaux étant physiquement représentés par deux niveaux différents de tension électrique.

Les fonctions électroniques de base travaillant en code binaire, on peut leur appliquer certains résultats mathématiques regroupés sous le nom d'Algèbre de Boole, ceci permettant des fonctions plus évoluées telles que l'addition binaire ou mieux l'addition décimale en utilisant les possibilités de transcodage.

Le code binaire étant d'un maniement relativement difficile par l'homme, on préfère utiliser des codes plus proches de la représentation décimale familière, tels que le code octal (base de numération 8) et le code hexadécimal (base 16), ces codes étant facilement déduits du binaire par la concaténation de, respectivement, 3 et 4 digits binaires. On dit alors par abus de langage que la machine travaille en octal ou en hexadécimal.

Cette première tentative de rapprochement entre la machine et l'homme n'est pas suffisante, en effet seul un informaticien très habitué est rompu à la gymnastique intellectuelle qui consiste à travailler en octal ou en hexadécimal.

Un autre palier facilitant les communications machine-homme, apparaît sous la forme d'un alphabet commun (alphabet étant pris au sens large, c'est-à-dire englobant aussi bien les chiffres que les lettres ainsi que tous les caractères dits spéciaux comme +, * , etc...), c'est-à-dire un jeu de caractères familiers à l'homme et facilement décodés par la machine. C'est cet alphabet qui est utilisé dans la conversation avec l'ordinateur par le biais de ses organes périphériques spécialisés pour les relations avec l'homme, c'est-à-dire essentiellement lecteur de cartes, télétype et imprimante.

Nous avons donc progressé en rendant les moyens de communications accessibles à tout un chacun sous la forme d'un ensemble de symboles familiers. Ceci n'est en fait qu'un petit pas, car si l'on a facilité la forme des communications, on n'a rien fait sur le contenu de celles-ci.

Il importe donc maintenant de cerner le contenu des conversations que peut avoir un informaticien avec un ordinateur. Nous savons qu'un ordinateur est généralement constitué d'une unité de commande, d'une unité de traitement, d'une mémoire centrale et de périphériques.

L'unité de commande prend en charge les ordres que par exemple nous avons inscrits en mémoire centrale par le biais de l'un des périphériques, il peut s'agir par exemple d'une addition, l'UC détectant alors l'instruction addition déclenchera l'opération en commandant l'unité de traitement.

Les ordres (ou instructions) font partie d'un ensemble plus ou moins grand appelé jeu d'instructions élémentaires qui regroupe sous un certain formalisme toutes les actions élémentaires pour lesquelles l'unité de traitement a été conçue. On les appelle couramment instructions-machine. Généralement une instruction-machine agit sur deux lots de données. Il peut s'agir, par exemple, d'additionner le contenu d'un premier registre à celui d'un second registre, ou de prélever des caractères se trouvant à une première adresse en mémoire centrale pour les envoyer à une seconde adresse.

En machine, on particularise les instructions par leur code-instruction (ou type-opération, ou code-opération) sous forme hexadécimale (dans la suite de ce mémoire, nous conserverons ce qualificatif, la machine utilisée ayant une représentation en base 16), chacune ayant bien sûr son code propre. Les opérandes, quant à eux, sont également sous forme hexadécimale, leur représentation variant selon le type d'adressage choisi pour la machine.

On voit donc maintenant que si l'on a la possibilité de disposer d'un ensemble d'opérandes voulus, il est possible d'effectuer un ensemble d'opérations plus ou moins complexe, en commandant une séquence d'instructions. On appellera programme cette séquence d'instructions.

Il existe des machines disposant d'un assembleur (auto-code) de bas niveau qui permet au programmeur d'écrire dans un langage d'assemblage où seules les étiquettes de renvoi sont symboliques, c'est le cas du GE 55 utilisé pour cette étude.

On voit bien que ce système est très contraignant puisqu'il oblige à coder "à la main" les instructions que l'on veut utiliser ainsi que les opérandes sur lesquels elles jouent.

Un système plus évolué apparaît avec les langages d'assemblage habituels, où codes-opérations et opérandes sont mnémoniques, le programme assembleur se chargeant de la transcription en langage machine. Ces langages d'assemblage sont susceptibles d'être utilisés par des programmeurs moins spécialisés que dans le cas précédent puisqu'ils n'ont à retenir que les noms symboliques des instructions. Néanmoins, ce genre de langage reste l'apanage de spécialistes, ceci étant dû, en partie à la structure du langage, d'un niveau assez bas puiqu'il faut décomposer une opération un peu complexe en une suite d'instructions élémentaires, en partie au fait que les assembleurs ne sont pas universels, chaque constructeur ayant son langage d'assemblage et très souvent même plusieurs types de langages d'assemblage selon les gammes de machines.

Il est donc apparu nécessaire de développer des langages universels, c'est-à-dire susceptibles d'être acceptés par tout ordinateur équipé pour les recevoir, et de niveau évolué, c'est-à-dire d'une souplesse d'utilisation telle qu'un non-spécialiste de l'informatique puisse rapidement apprendre un petit nombre de règles précises lui permettant de programmer facilement dans le champ d'application qui lui est propre.

De même qu'un programme spécialisé appelé assembleur prend en charge un programme écrit en langage d'assemblage, il y a un programme spécialisé capable de prendre en charge un programme écrit dans un langage évolué. Ce programme spécialisé s'appelle un compilateur.

Il existe, pour une même machine, autant de compilateurs que de langages évolués qu'on veut pouvoir utiliser ; à titre d'exemples, citons : FØRTRAN, CØBØL, ALGØL, PL1, APL, etc...

Le langage évolué qu'on veut utiliser ici est le BASIC (Beginner's All-Purpose Symbolic Instruction Code) et nous ferons l'étude du compilateur associé sur GE 55.

Le premier chapitre montre les propriétés des compilateurs et établit les comparaisons entre leurs grands types.

Le second chapitre traite des langages et des grammaires en faisant état des travaux des linguistes qui, tel N. CHOMSKY, ont grandement contribué à l'essai de formalisation des langages naturels.

Le troisième chapitre fait la description du système informatique dont nous nous sommes servis dans le cadre de cette étude, aussi bien du point de vue hardware, que du point de vue software.

Le chapitre IV revient sur les langages en exposant les différentes méthodes d'analyse syntaxique attachées aux langages de programmation.

Dans le chapitre V, nous donnons les règles de grammaire du langage BASIC, sur lesquelles nous faisons fonctionner l'analyseur syntaxique. Ce chapitre décrit également le fonctionnement de cet analyseur.

L'interpréteur, quant à lui, fait l'objet du sixième chapitre qui décrit en détail tous les mécanismes employés pour l'interprétation des variables simples, des matrices, des fonctions, des quantités numériques, etc... On y trouve aussi le détail des mécanismes de traitement de tous les ordres qui constituent le BASIC.

Le septième chapitre expose la mise en oeuvre pratique du compilateur réalisé dans le cadre du système utilisé. C'est en fait le mode d'emploi du BASIC sur GE 55.

Enfin, des exemples probants de programmes BASIC mis en oeuvre sur GE 55 figurent au chapitre VIII.

CHAPITRE I

COMPILATEURS

I-1 - TRADUCTEURS

Nous venons de voir qu'un compilateur est un programme qui assure la traduction d'un programme écrit dans un langage évolué en un programme que la machine peut comprendre et exécuter.

Pour remplir ce rôle, on peut imaginer diverses méthodes. Soit $\mathcal{P}_L$ un programme écrit dans un langage L, et soit $\mathcal{P}_M$ le même programme écrit en langage machine M. Il faut en fait réaliser le passage ou la traduction de $\mathcal{P}_L$ en $\mathcal{P}_M$.

Sachant que l'on dispose d'un traducteur $\mathcal{C}_M^{A \to M}$ du langage d'assemblage (noté A) en langage machine (c'est le programme assembleur), on peut évidemment penser à la démarche suivante : passer de $\mathcal{P}_L$ en $\mathcal{P}_A$ par un traducteur $\mathcal{C}_M^{L \to A}$, traducteur de L en A écrit en langage machine M, on peut noter ce passage :

$$\mathcal{C}_M^{L \to A} \quad \mathcal{P}_L \longrightarrow \mathcal{P}_A$$

puis utiliser l'assembleur :

$$\mathcal{C}_M^{A \to M} \quad \mathcal{P}_A \longrightarrow \mathcal{P}_M$$

On a bien alors obtenu $\mathcal{T}_M$ exécutable sur la machine M. Cette méthode n'est pas fréquemment utilisée. En effet, si elle simplifie le travail de compilation en laissant à l'assembleur le soin de réserver les variables et les tableaux en machine, ainsi que le soin de produire des programmes acceptables par l'éditeur de liens, en revanche le temps séparant l'introduction de $\mathcal{T}_L$ et l'exécution de $\mathcal{T}_M$ devient vite suffisamment prohibitif pour que l'on réserve cette méthode à la mise au point de compilateurs expérimentaux. D'autre part, cette méthode n'est pas applicable à un compilateur interprétatif (dont on verra la signification bientôt) du type de celui que nous nous proposons de construire.

On emploiera donc un traducteur direct $\mathcal{C}_M^{L\to M}$ tel que :

$$\mathcal{C}_M^{L\to M} \quad \mathcal{T}_L \to \mathcal{T}_M$$

Remarque :

Nous passons sous silence les traducteurs de langages évolués entre eux, ainsi que les traducteurs travaillant dans le langage d'une machine qui produisent des langages exécutables sur une autre machine, le compilateur que nous étudions ici étant particularisé à la seule machine utilisée : le GE 55.

Définissons maintenant compilateurs génératifs et compilateurs interprétatifs.

I-2 - COMPILATEUR GENERATIF

On appelle couramment programme-source le programme présenté à l'entrée de l'ordinateur, langage-source le langage dans lequel ce programme est écrit, et langage-cible le langage dans lequel est écrit le programme issu du compilateur.

Un compilateur génératif produit un programme machine susceptible d'être placé sur un support convenable permettant de multiples utilisations, ce programme machine est encore appelé programme binaire ou "deck".

Dans la suite de l'exposé, nous oublierons souvent le mot programme, étant bien entendu que c'est le même programme qui va de l'entrée à l'exécution en passant par différentes étapes.

On peut schématiser le passage du programme en machine de la façon suivante :

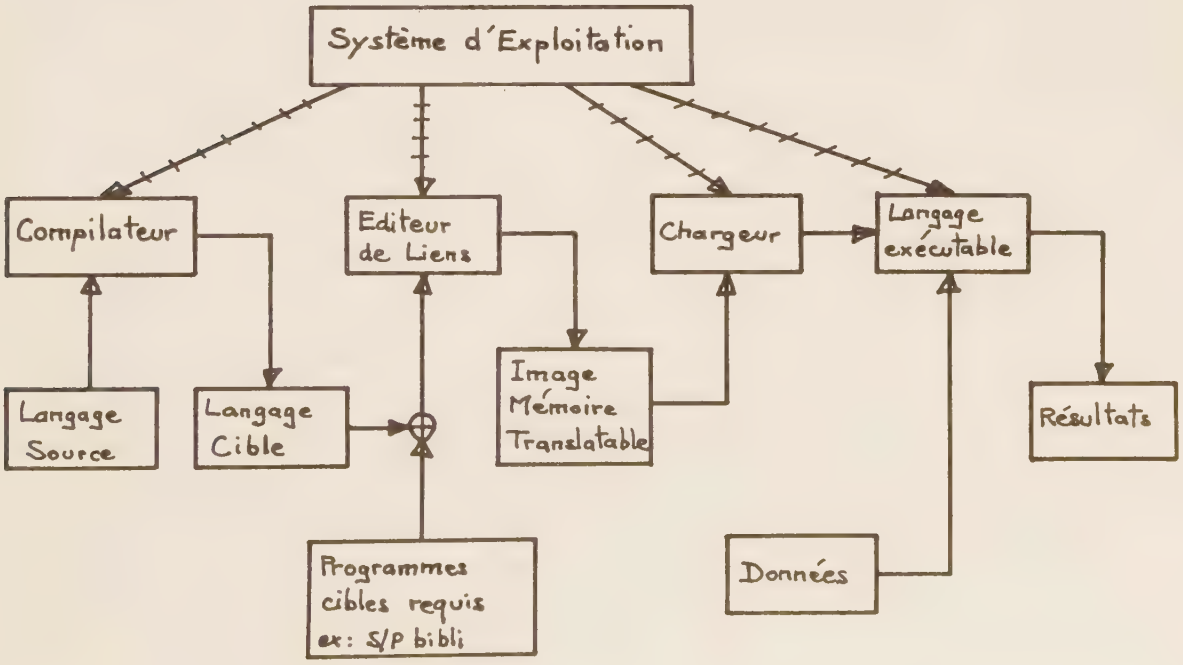


Figure I-1

En cas de nouvelle utilisation du programme, il suffit de réutiliser le langage-cible, la phase compilation n'ayant plus de raison d'être.

I-3 - COMPILATEUR INTERPRETATIF

Un compilateur interprétatif produit un programme-cible exécuté au fur et à mesure de la progression de la traduction. Il va de soi que par étapes la traduction s'arrête pour laisser place à l'exécution.

On peut donner le schéma suivant pour le passage en machine :

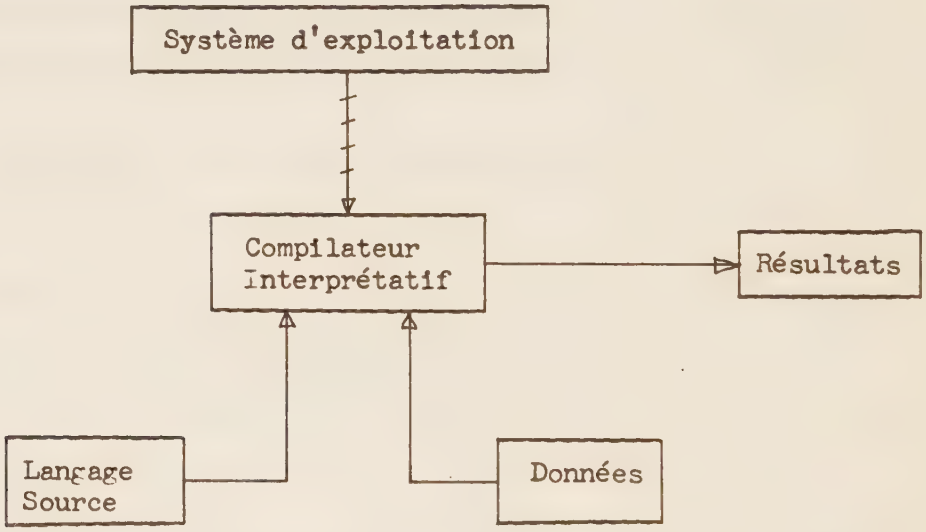


Figure I-2

En fait, ce schéma est très simplifié. En réalité, un compilateur interprétatif prend en charge un langage le plus souvent conversationnel, ce terme venant du fait que le programme s'exécute sous le contrôle de l'opérateur assis devant un télétype. C'est ainsi

que l'opérateur est amené à entrer au clavier du télétype des informations qui peuvent être soit des données, soit des indications qui, prises en charge par le programme, commanderont son évolution future.

Généralement, le langage-source est lui-même introduit en machine ligne à ligne au moyen du télétype. Les résultats fournis par le programme sont transmis à l'opérateur de la même façon. C'est une véritable conversation qui s'instaure.

Remarquons que la figure I-2 ne fait pas apparaître de langage-cible, ce qui signifie qu'en mode interprétation, il n'est pas question de faire une seule compilation suivie de plusieurs passages du module-cible, mais qu'il faut interpréter le langage-source autant de fois que l'on veut passer le programme.

I-4 - REMARQUES

Un rôle très important dont nous parlerons dans les chapitres suivants, que doivent remplir les compilateurs, consiste à faire l'analyse syntaxique du langage-source pour vérifier la bonne écriture du programme et ce avant toute génération ou interprétation.

I-5 - COMPARAISONS

Un générateur a l'avantage précieux sur un interpréteur de produire un module-cible réutilisable à la demande et performant, l'interpréteur ne fournissant pas un produit réutilisable. Le même type d'avantage apparaît dans le traitement des séquences itératives, le générateur produisant une séquence itérative en langage-cible, l'interpréteur devant reprendre en charge chaque instruction de la boucle autant de fois que l'on veut itérer.

Les avantages de l'interpréteur apparaissent dans l'analyse syntaxique d'un langage conversationnel ; en effet, on peut bénéficier de la structure ligne à ligne du programme pour faire l'analyse syntaxique au fur et à mesure que les lignes sont entrées, une ligne mal écrite syntaxiquement ne remettant pas en cause le travail précédent ; il suffit de retaper cette ligne, et l'on reprend son analyse. En cas d'erreur syntaxique, un générateur perd, quant à lui, tout le bénéfice du travail produit jusqu'à la détection de l'erreur. Il faut corriger l'erreur et recommencer la compilation. Enfin, le gros avantage de l'interpréteur reste le dialogue programme-opérateur.

Globalement, on peut dire que les générateurs sont plus performants et que les interpréteurs sont plus souples.

Le BASIC étant un langage conversationnel, nous sommes amenés à choisir un compilateur interprétatif.

CHAPITRE II

LANGAGES-GRAMMAIRES

II-1 - GENERALITES

Les langages utilisés en informatique, dits langages de programmation, sont des langages artificiels généralement très proches du langage mathématique, donc familiers à un mathématicien ou à un physicien (du moins, les langages orientés vers la programmation mathématique), et, par certains aspects, proches aussi du langage naturel.

Il faut disposer de règles précises caractérisant le langage employé.

Remarquons que les expressions (ou mots) apparaissant dans un langage de programmation se présentent sous la forme de chaînes de caractères (concaténations de caractères), ce qui facilite leur analyse, car nous pourrons considérer les caractères en séquence.

Les règles de formation permettant de générer les phrases correctes du langage définissent la syntaxe, les règles qui permettent d'attribuer une signification aux mots définissent la sémantique du langage.

II-2 - LANGAGES

Un langage est formé de mots constitués par des lettres appartenant à un alphabet. On dit aussi chaîne pour mot et symbole pour lettre. Donnons maintenant quelques définitions.

Alphabet ou Vocabulaire Terminal : V_T, c'est un ensemble non vide et fini d'éléments.

Symboles de base : ce sont les éléments de V_T

Chaîne : une séquence de symboles de base concaténés forme une chaîne

Exemple : $V_T = \{1, 2\}$

Exemple de chaînes qu'on peut former sur V_T :

1 ; 2 ; 12 ; 21 ; 11 ; 22 ; 12212 ; etc...

Chaîne vide : notée Λ , elle ne contient pas de symbole

Longueur d'une chaîne : notée /chaîne/, elle représente le nombre de symboles contenus dans cette chaîne.

Exemple : /11/ = 2 ; /12212/ = 5

V_T^* = c'est l'ensemble des chaînes formées sur V_T, c'est un ensemble infini dénombrable .

Un langage défini sur V_T est une partie L de l'ensemble V_T^* .

Un langage vide est un langage dont la seule chaîne est Λ .

Un langage fini contient un nombre fini de chaînes.

Un langage infini est un langage non vide et non fini.

On appelle Monoïde tout ensemble muni d'une loi de composition interne associative et ayant un élément neutre (la différence avec un groupe apparaît dans la non-existence d'un élément symétrique).

Dans notre cas, l'ensemble V_T^* est muni d'une loi de composition interne, la concaténation.

Soient les chaînes a et b $\in V_T^*$ $\quad ab \in V_T^*$ où $a = a_{i_1} a_{i_2} \ldots a_{i_u}$ et $b = b_{i_1} b_{i_2} \ldots b_{i_v}$

En concaténant les chaînes a et b, on obtient :

$$ab = c = a_{i_1} a_{i_2} \ldots a_{i_u} b_{i_1} b_{i_2} \ldots b_{i_v}$$

avec /c/ = /a/ + /b/

L'opération concaténation est associative, en effet :

$$a(bc) = (ab)c = abc$$

La chaîne vide est l'élément neutre :

$$a \Lambda = \Lambda a = a$$

L'ensemble V_T^* a donc une structure de monoïde.

Comme il n'existe pas d'autres relations que l'associativité et l'élément neutre, le monoïde est dit libre.

Le monoïde libre V_T^* a V_T pour générateur.

$$V_T^* = V_T^0 \cup V_T^1 \cup V_T^2 \cup \ldots \cup V_T^n \cup \ldots$$

où $V_T^0 = \{\Lambda\}$ = chaîne vide

$V_T^1 = V_T$ chaînes de longueur 1

$V_T^n = V_T \, V_T^{n-1}$ chaînes de longueur n

On dit qu'un langage est formalisé si l'on peut le définir par un ensemble de chaînes appartenant à V_T^* , par un élément distingué, appelé racine ou axiome, et par un ensemble fini de règles destinées à le décrire. Ces règles donnent par remplacements successifs à partir de l'axiome, toutes les chaînes bien formées du langage susceptibles d'être obtenues.

II-3 - GRAMMAIRES

Les règles que nous venons d'évoquer constituent les règles de la grammaire. Pour écrire ces règles, nous verrons au § II-5 que l'on peut employer un langage spécial, appelé méta-langage, construit à l'aide d'un vocabulaire de méta-symboles.

Pour définir les règles de grammaire, on est amené à classer les éléments terminaux en diverses catégories syntaxiques, qui elles-mêmes sont regroupées dans des catégories de niveau supérieur, etc..., l'ensemble de ces catégories syntaxiques constituant le vocabulaire non terminal V_N.

Résumons ici un certain nombre de conventions :

V_T = vocabulaire terminal = { symboles de base }

V_N = vocabulaire non terminal = { éléments syntaxiques }

$V = V_T \cup V_N$ = vocabulaire total

$V_T \cap V_N = \emptyset$ V_N est disjoint de V_T

Nous noterons par des petites lettres les chaînes sur V_T
par des grandes lettres les chaînes sur V_N
par des lettres grecques les chaînes sur V

Soit V^* l'ensemble des chaînes sur V.

Considérons la relation binaire $\varphi \Longrightarrow \psi$ dans V
qui définit une production et qui se lit : " φ se réécrit ψ ".

La grammaire est dite de CHOMSKY si la relation satisfait aux conditions suivantes :

I°/ - La relation $\Longrightarrow$ n'est pas réflexive

2°/ - $A \in V_N$ si et seulement si $\exists\, \varphi, \psi, \omega$ tels que

$$\varphi\, A\, \psi \Longrightarrow \varphi\, \omega\, \psi$$

3°/ - $\nexists\, \varphi, \psi, \omega$ tels que :

$\varphi \Longrightarrow \psi \# \omega$ où $\#$ est un élément séparateur

$$\# \notin V_T, V_N$$

$$V = V_T \cup V_N \cup \{\#\}$$

4°/ - Il existe un ensemble de couples de mots :

$$(\alpha_1, \beta_1), \ldots (\alpha_n, \beta_n)\ (\alpha_i \in V^* - \Lambda, \beta_i \in V^*\ \forall_i\ 1 \leq i \leq n)$$

tels que $\forall\, \varphi$ et ψ , on puisse écrire $\varphi \Longrightarrow \psi$
si et seulement si $\exists\, \varphi_1$, et φ_2 et $1 \leq j \leq n$ tels que

$$\varphi = \varphi_1 \alpha_j \varphi_2 \quad \text{et} \quad \psi = \varphi_1 \beta_j \varphi_2$$

(φ_1 et $\varphi_2 \in V^*$, peuvent donc être vides)

On peut parmi les α_i, en avoir certains qui sont égaux.

Exemple : Soit $V = V_T \cup V_N \cup \{\#\}$

où $V_T = \{a, b, c\}$

$V_N = \{S, A, B, C\}$

Soit un ensemble de couples de mots :

$\{(S, AbB), (A,SA), (A,bBC), (bBAb, bCB), (\#Ab, \#Cb), (bB,bac), (C,a)\}$

Soit la chaîne $\varphi = SAbB$.

Selon φ_1 et φ_2, on sera amené à choisir une règle ou une autre. Ainsi :

si $\varphi_1 = S$ et $\varphi_2 = bB$, $\alpha = A$ et si l'on prend le couple (A,bBC), il vient $SAbB \Longrightarrow SbBCbB$

si $\varphi_1 = \Lambda$ et $\varphi_2 = AbB$, $\alpha = S$ et si l'on prend le couple (S, AbB) (remarque : pour $\alpha = S$, c'est le seul couple qu'on peut prendre), il vient : $SAbB \Longrightarrow AbBAbB$.

On dit aussi que l'opération $\varphi \Longrightarrow \psi$ est une dérivation. Définissons une dérivation composée, comme étant un enchaînement de dérivations directes.

Par exemple, soit une suite finie $\alpha_1, \alpha_2, \ldots \alpha_n$ de chaînes sur V^*, telle que chaque α_i dérive du précédent.

$\alpha_{i-1} \Longrightarrow \alpha_i$, en écrivant la suite des dérivations

$\alpha_1 \Rightarrow \alpha_2$, $\alpha_2 \Rightarrow \alpha_3$, $\ldots$, $\alpha_{n-1} \Longrightarrow \alpha_n$, on définit finalement la dérivation composée $\alpha_1 \overset{*}{\Longrightarrow} \alpha_n$.

Reprenons l'exemple déjà vu, et dérivons S :

$S \Longrightarrow AbB \Longrightarrow SAbB \Longrightarrow AbBAbB \Longrightarrow AbCBB...$
ou bien $S \overset{*}{\Longrightarrow} AbCBB$

Remarquons que si nous avons défini pour la grammaire ses vocabulaires et les règles de production (ou de réécriture), nous n'avons pas précisé à partir de quel élément il convenait d'appliquer les règles. Il faut alors distinguer dans V_N un élément privilégié qu'on appelle axiome (noté S).

On peut maintenant définir une grammaire comme étant la donnée du quadruplet :

$$G = (V_T, V_N, S, P)$$

où V_T, V_N et S ont la signification habituelle et où P est l'ensemble des règles de production.

Remarque : $P \subset \{V^* - \Lambda\} \times \{V^*\}$ puisque les règles sont de la forme $\varphi \Longrightarrow \psi$ où $\varphi \in \{V^* - \Lambda\}$
$\psi \in \{V^*\}$

Le langage L défini sur la grammaire G (ou engendré par la grammaire G) est le sous-ensemble de V_T^* des chaînes c telles que $S \overset{*}{\Longrightarrow} c$

$$L(G) = \left\{ c \ / \ S \overset{*}{\Longrightarrow} c \ \text{ et } c \in V_T^* \right\}$$

Remarque importante :

Il est bien clair que la donnée de la grammaire seule n'est pas suffisante pour obtenir les mots du langage, il faut de plus se donner l'algorithme qui permet d'appliquer les productions qui, à partir de S, donneront les mots de L

II-4 - CLASSIFICATION DES GRAMMAIRES

CHOMSKY a proposé la classification des grammaires en quatre types.

II-4-1 - Type 0

Une grammaire satisfaisant aux 4 conditions de CHOMSKY est dite de type 0 ainsi que le langage correspondant.

II-4-2 - Type 1

Si la grammaire satisfait aux 4 conditions de CHOMSKY et si les règles $\varphi \Longrightarrow \psi$ sont telles que :

$$\varphi = \alpha A \beta \qquad , \ \alpha, \beta \in V^* \text{ et } A \in V_N$$

$$\psi = \alpha \omega \beta \qquad \omega \in V^* - \Lambda$$

on dit que la grammaire est de type 1 ou grammaire à dépendance contextuelle (context-sensitive), ce qui signifie que A se réécrit ω dans le contexte α, β .

II-4-3 - Type 2

Les 4 conditions de CHOMSKY étant remplies, si les règles $\varphi \Longrightarrow \psi$ sont telles que :

$$\varphi = A \qquad A \in V_N$$

$\psi = \omega \qquad \omega \in V^* - \Lambda$, la grammaire est dite de type 2 ou à indépendance contextuelle (context-free). On dit aussi que les grammaires de ce type sont des C-grammaires et les langages correspondants, des C-langages.

II-4-4 - Type 3

Si les règles $\varphi \longrightarrow \psi$ sont de la forme :

$$\begin{cases} \varphi = A \\ \psi = a \text{ ou } aB \end{cases} \quad \text{ou} \quad \begin{cases} \varphi = A \\ \psi = a \text{ ou } Ba \end{cases} \qquad A, B \in V_N^* \qquad a \in V_T^*$$

et si les 4 conditions de CHOMSKY sont remplies, la grammaire est dite de type 3, d'états finis ou régulière.

Une grammaire de ce type est également appelée K-grammaire (grammaire de KLEENE) et le langage correspondant, K-langage.

On voit que pour une grammaire de ce type, chaque règle de production fait apparaître à droite au moins un élément terminal.

II-5 - EXEMPLE - REPRESENTATION DE BACKUS-NAUR (BACKUS-NAUR-FORM - BNF)

Soit P l'ensemble des règles de grammaire suivantes :

R1 <nombre décimal> ::= <entier> | <entier pointé><entier> | <entier pointé> | <point><entier>

R2 <entier pointé> ::= <entier><point>

R3 <entier> ::= <chiffre> | <entier><chiffre>

R4 <point> ::= .

R5 <chiffre> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Précisons le méta-langage employé ici :

::= est le symbole de réécriture, il remplace $\Longrightarrow$

<xxx> symbolise un élément non-terminal

| symbolise une alternative, il sépare les alternants

par exemple <entier> se réécrit soit <chiffre> , 1er alternant, soit <entier> concaténé à <chiffre> , second alternant.

L'axiome est ici <nombre décimal>.

Cherchons la dérivation composée qui à partir de l'axiome nous amènera à la chaîne terminale suivante 12.345

<nombre décimal> $\xrightarrow{R1}$ <entier pointé><entier> $\xrightarrow{R2}$
<entier><point><entier> $\xrightarrow{R3}$ <entier><chiffre><point><entier>

$\xrightarrow{R3}$ <chiffre><chiffre><point><entier>

$\xrightarrow{R3}$ <chiffre><chiffre><point><entier><chiffre>

$\xrightarrow{R3}$ <chiffre><chiffre><point><entier><chiffre><chiffre>

$\xrightarrow{R3}$ <chiffre><chiffre><point><chiffre><chiffre><chiffre>

$\xrightarrow{R4/5}$ 1 2 . 3 4 5

On a trouvé une dérivation composée permettant d'obtenir la chaîne terminale souhaitée, cette dernière appartient bien alors au langage, on peut dire qu'elle est bien formée (chaîne bien formée : CBF).

Remarquons que la règle R3 définit une récursivité à gauche <entier> ::= <entier><chiffre> , il existe deux autres types de récursivité ; la récursivité à droite, et l'auto-enchâssement (self-embedding) qui se symbolisent respectivement par :

$$X \longrightarrow \gamma\ X$$

et $X \longrightarrow \alpha\ X\ \beta$ dans les notations de CHOMSKY.

II-6 - GRAPHE DE DERIVATION - ARBRE SYNTAXIQUE

On peut faire correspondre un graphe de dérivation à la structure d'une chaîne appartenant à un langage de type 2 ou 3.

Reprenons l'exemple précédent, le graphe se présente sous la forme :

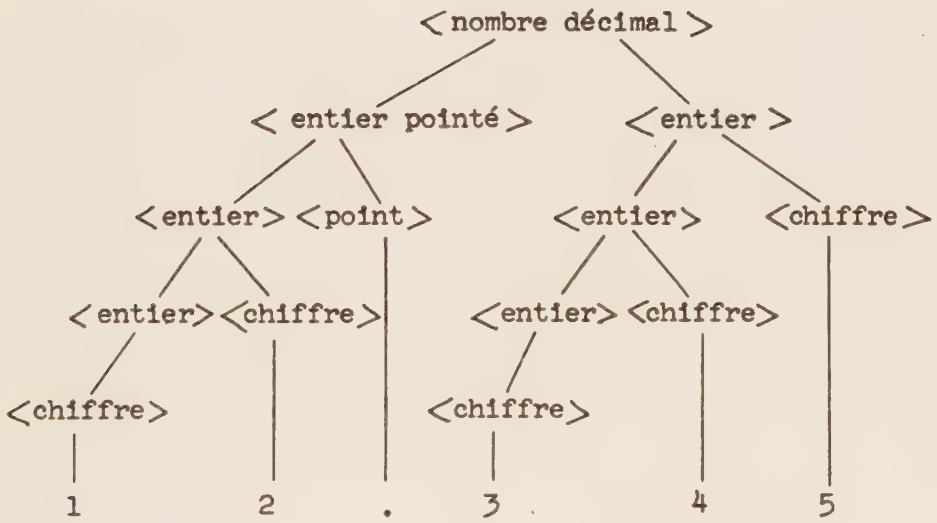


Figure II-1

Une grammaire est dite ambiguë s'il existe au moins deux structures différentes pour une même chaine terminale, ce qui se traduit, sous une autre forme, par deux graphes différents.

Exemple de grammaire ambiguë :

$$G = (V_T, V_N, S, P)$$

$$V_T = \{ a \} , \quad V_N = \{ S \}$$

$$P = \begin{cases} S \longrightarrow aS \\ S \longrightarrow aSa \\ S \longrightarrow a \end{cases}$$

Pour la chaîne terminale aaaa, on trouve les deux graphes suivants :

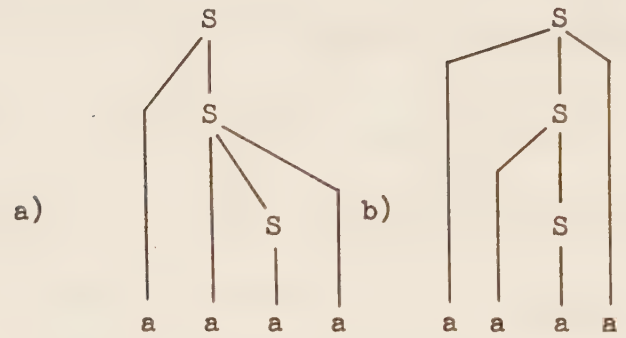


Figure II-2

Remarquons que les éléments composant le graphe de dérivation proviennent de $V_T \cup V_N$, les feuilles de l'arbre appartiennent à V_T, les noeuds appartiennent à l'ensemble des éléments syntaxiques, les branches traduisent les règles employées, c'est pourquoi le graphe de dérivation s'appelle couramment arbre syntaxique.

C'est l'axiome qui constitue l'élément racine de l'arbre.

II-7 - ANALYSE SYNTAXIQUE

En pratique, quand on veut traduire un programme, il importe de savoir si les chaînes de caractères qui composent ce programme sont bien formées, ce qui revient à vérifier si elles dérivent de l'axiome selon les règles syntaxiques en vigueur dans la grammaire du langage utilisé.

Ce processus correspond à la phase d'analyse syntaxique. Les diverses méthodes d'analyse et leur discussion font l'objet d'un chapitre particulier.

CHAPITRE III

DESCRIPTION DE LA MACHINE

III-1 - CONFIGURATION

L'ordinateur utilisé dans le cadre de cette étude est le modèle GE 55 de la Compagnie Honeywell-Bull. Nous l'avions à notre disposition dans la configuration suivante :

- Unité centrale avec mémoire centrale de 10 000 octets.
- Lecteur de cartes code Hollerith - 150 cartes/mn
- Perforateur de carte - 40 col/s
- Imprimante 140 l/mn
- Tambour magnétique comportant 128 pistes de 700 octets
- Clavier alpha-numérique permettant l'introduction directe de caractères en mémoire centrale
- Clavier numérique associé à une unité de visualisation de 6 caractères numériques

III-2 - SOFTWARE

Trois softwares de base sont disponibles sur cette machine, deux systèmes-cartes et un système-tambour. Le système-carte BOS/C (Basic Operating System/Cards) permet de travailler avec un assembleur

auto-code sur l'ensemble de la mémoire et d'utiliser le tambour comme support de fichiers.

Le système-carte EØS/C (Extended Operating System/Cards) permet de travailler avec un assembleur de type symbolique, programmation plus souple donc, mais mise en oeuvre plus pénible.

Le système-tambour BØS/MD (Basic Operating System/Magnetic Drum) offre l'avantage de permettre l'exécution de programmes importants qui dépassent la taille de la mémoire centrale en utilisant la technique du recouvrement (overlay) ; l'assembleur utilisé est de type auto-code.

Un compilateur MINIFØRTRAN peut être utilisé dans le cadre BØS/MD.

Dans chacun de ces systèmes, on dispose d'un certain nombre de sous-programmes fournis par le constructeur, en particulier les sous-programmes mathématiques.

Le but de cette étude est d'implanter un compilateur BASIC dans le cadre BØS/MD.

III-3 - UNITE CENTRALE

Il est intéressant de donner quelques informations significatives sur l'unité centrale, ces informations nous aideront à voir les contraintes qu'impose le système utilisé.

L'unité centrale assure la gestion du système à l'aide d'un organe de commande et d'une mémoire interprête. Elle exécute les opérations requises grâce aux opérateurs élémentaires associés à la mémoire interprête, les programmes et les données à traiter étant en mémoire centrale.

III-3-1 - La mémoire centrale

Il est important de détailler ce qui se rattache à la mémoire centrale, puisqu'en définitive, le programmeur n'est en rapport qu'avec elle.

La mémoire centrale à notre disposition se présente sous la forme de 10 000 positions dans une technologie à tores magnétiques. Chaque position est constituée de 9 digits binaires dont 8 sont utilisables et constituent un octet, le 9ème étant un bit d'imparité. Chaque octet est directement adressable, l'adresse variant de 0000 à 9999.

La mémoire est organisée en zones pour assurer le déroulement convenable de tous les échanges d'informations.

III-3-1-1 - Zone de communication

Utilisée par l'organe de commande, cette zone sert à gérer l'enchaînement des programmes (pseudo-multiprogrammation de 5 programmes en machine) et le fonctionnement des périphériques.

Implantée en bas de mémoire, elle occupe 85 octets de l'adresse 0000 à l'adresse 0084.

III-3-1-2 - Zone des registres spécialisés

Les registres spécialisés sont de deux types : les registres-tests et les registres-programmes. Cette zone occupe 11 octets (0085 à 0095).

Les registres-tests fournissent des informations mises en place par certaines instructions. On trouve le registre-signe qui nous donne le signe d'un nombre, le registre-comparaison qui nous indique le résultat de la comparaison de deux opérandes, le registre-

dépassement de capacité positionné par les instructions de calcul numérique.

Les registres-programmes contiennent des adresses sous la forme de 4 chiffres décimaux condensés dans 2 octets. Le plus important est le registre d'adresse-programme évoluant à chaque instruction et qui contient à tout instant l'adresse de la prochaine instruction à exécuter.

III-3-1-3 - Zone des registres numériques

On verra bientôt qu'une quantité numérique en format standard occupe 5 octets en mémoire. Un registre numérique est alors une succession de 5 octets en mémoire vive affectée d'une adresse fixe. On peut disposer de 100 registres numériques numérotés de 00 à 99, occupant 500 octets en mémoire (0096 à 0595). Les registres 00 à 09 sont spécialisés comme registres de base, ils contiennent l'adresse origine de zones en mémoire banale, sous la forme de 4 chiffres décimaux condensés dans leurs 2 octets de droite.

III-3-1-4 - Mémoire banale

C'est toute la partie de la mémoire centrale qui n'a pas été spécialisée, elle comporte tous les octets compris entre les adresses 0596 et 9999.

Elle contient les programmes à exécuter, leurs fichiers de données ou de résultats, et les zones entrées/sorties comportant les informations à échanger entre la mémoire centrale et les périphériques.

Elle contient aussi, tout ou partie du superviseur, en haut de mémoire (zone de longueur variable selon le système utilisé).

III-3-1-5 - Représentation des informations en mémoire centrale

Il existe typiquement 2 représentations en mémoire centrale. La première est la représentation en code caractère. Chaque caractère occupe un octet. Si l'on découpe un octet en deux quartets, chaque quartet contient un chiffre hexadécimal. On l'appelle aussi représentation étendue. La seconde représentation est dite condensée, il y a encore deux chiffres hexadécimaux par octet, mais leur réunion ne correspond plus au code caractère.

Les nombres algébriques sur lesquels on peut exécuter des opérations de calcul, doivent être en représentation condensée.

La représentation numérique est décimale. Un nombre occupant 5 octets, on dispose de 10 chiffres décimaux, on spécialise le chiffre de plus haut poids pour indiquer le signe :

0 pour un nombre positif ou nul,

9 pour un nombre négatif.

Un nombre négatif est représenté par son complément à 10^{10},

par exemple + 5 est représenté | 0 | 000000005

- 5 est représenté | 9 | 999999995

↑ signe

Remarquons que l'on doit travailler dans certains cas sur des nombres en double longueur représentés par 19 chiffres décimaux, plus le signe.

Remarque :

Le code caractère en mémoire centrale n'est pas toujours le même que le code caractère externe selon les périphériques utilisés. Cela nécessite alors une opération de transcodage sur le code caractère.

III-3-1-6 - Mécanismes d'adressage en mémoire vive

Rappelons que l'adresse réelle d'un octet en mémoire est sous forme décimale (entre 0000 et 9999).

Si on définit une zone ou un mot en mémoire centrale, on peut avoir besoin soit de l'adresse gauche (en fait celle de l'octet qui précède la zone ou le mot), soit de l'adresse droite, selon le type d'instruction qu'on utilise. (L'adresse droite est supérieure à l'adresse gauche).

III-3-1-6-1 - Adressage direct

Dans une instruction de branchement, après assemblage, l'adresse de branchement se présente sous la forme d'une adresse absolue (ou réelle) exprimée par 4 chiffres décimaux dans 2 octets.

Un autre type d'adresse directe permet d'accéder aux registres numériques, il suffit de préciser leurs numéros dans les instructions. Remarquons que dans certains cas, on peut avoir besoin de l'adresse réelle (de l'octet de droite) d'un registre. Les registres étant implantés en mémoire centrale, on connaît leurs adresses par la formule Ad = 5N + 100 où N est le numéro du registre considéré.

III-3-1-6-2 - Adressage basé

Dans certains cas, les adresses des différentes zones sont définies par leur éloignement d'une adresse prise pour origine.

L'adresse origine s'appelle la base, et l'éloignement s'appelle déplacement.

La base est en fait le contenu d'un registre appelé registre de base (c'est l'un des registres 00 à 09).

Adresse réelle = (Registre de base) + Déplacement
(R) signifie contenu de R.

Dans une instruction, on rencontre ce type d'adressage sous la forme de 2 octets |Bx|xx| .

B est le numéro du registre de base (0 à 9).
xxx est le déplacement 000 à 999.

III-3-2 - Mémoire interprète

La mémoire interprète (appelée aussi mémoire morte) se présente sous la forme de 1024 mots de 36 bits. Elle contient, sous forme inaltérable, toutes les informations fixes, micro-programmes et tables, qui définissent le processus d'exécution de toutes les instructions utilisables.

Le cycle d'extraction et d'exécution des micro-instructions est de 7,9 µs, la durée d'une instruction est donc un multiple de ce cycle.

Les micro-fonctions sont exécutées par des opérateurs élémentaires appelés micro-opérateurs travaillant exclusivement sur des chaînes binaires dont la plus longue est l'octet.

La mémoire interprète est inaccessible au programmeur.

III-4 - INSTRUCTIONS

Les instructions du GE 55 sont au nombre de 67 réparties en 11 grandes catégories :

1 - Transferts internes
2 - Opérations arithmétiques
3 - Décalages
4 - Comparaisons
5 - Instructions logiques
6 - Variantes (branchements)
7 - Transferts entrée-sortie
8 - Traductions
9 - Mise au point de programme
10 - Multiprogrammation
11 - Connexions d'éléments périphériques

Toutes ces instructions sont microprogrammées sauf une : la division qui doit être effectuée par un sous-programme.

Remarquons qu'il n'y a pas d'unité arithmétique et que les opérations arithmétiques sont effectuées par consultation des tables insérées en mémoire morte.

Donnons pour fixer les idées, quelques temps opérations :

Instruction NOP non opération : 0,35 ms
Instruction MRR transfert de registre à registre : 1,5 ms
Instruction MPD Multiplication : 14,42 + 6,28 C + A ms

$A = 2{,}42$ ms si $m > 0$ et $M < 0$
$A = 1{,}60$ si $m < 0$, $\forall M$

m = multiplicateur, C son nombre de chiffres significatifs
M = multiplicande

Instruction DVD Division : 133,6 + 12,4 x N ms

N = nombre de chiffres du quotient

Ces quelques exemples montrent que l'on peut obtenir très rapidement des temps d'exécution de programme importants, mais nous verrons plus loin qu'il y a d'autres causes qui augmentent les temps d'exécution.

III-5 - BASIC OPERATING SYSTEM/TAMBOUR

Il est intéressant de donner diverses informations relatives au système d'exploitation Tambour que nous utiliserons. En effet, il est évident que dans la place disponible en mémoire centrale, environ 8 000 octets, compte tenu des zones réservées et du système résident, il est impossible de faire tenir un compilateur, un programme BASIC à exécuter avec des tables qui peuvent être importantes, sachant que le BASIC est un langage disposant d'instructions matricielles.

Nous ne pouvons donc pas nous contenter d'un système réduit, et nous allons être obligés d'utiliser toutes les ressources que nous donne le système d'exploitation en tambour.

La première ressource intéressante qui nous est offerte, consiste en la segmentation des programmes. C'est-à-dire qu'on peut découper un programme de taille importante, en différents segments qui se succéderont en mémoire centrale, lors de l'exécution. C'est la technique du recouvrement (overlay). On peut dans certains cas, utiliser des pseudo-instructions spécifiques à cette technique, pour orienter la segmentation de façon judicieuse en interdisant par exemple la segmentation d'une séquence itérative.

L'utilisation de pistes banalisées du tambour, pour enregistrer le programme BASIC à exécuter, ainsi que ses tables et divers renseignements utiles au compilateur, permet de libérer la mémoire centrale et de la réserver aux modules dont on a besoin en permanence.

L'emploi de macro-instructions de transfert entre tambour et mémoire, facilite la programmation, et nous avons cette possibilité grâce à l'emploi du BOS/MD.

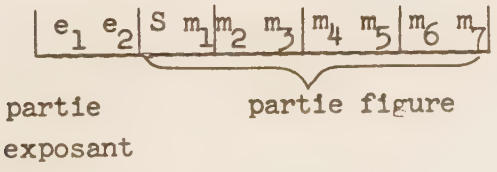
Dans ce système d'exploitation, le moniteur-superviseur et l'assembleur sont eux-mêmes segmentés pour occuper le minimum de mémoire possible, et cela est particulièrement important pour le moniteur-superviseur dont une partie doit toujours résider en mémoire lors de l'exécution d'un programme.

On peut dire que l'utilisation du tambour dans ce système d'exploitation étend artificiellement la taille de la mémoire centrale. Tout se passe comme si l'on avait une mémoire plus grande. C'est la technique de la mémoire virtuelle.

III-6 - REPRESENTATION FLOTTANTE

Le BASIC travaillant avec des variables réelles et disposant de fonctions internes également réelles, il faut utiliser la représentation flottante.

Un nombre en représentation flottante se présente sous la forme d'une figure et d'un exposant : 0, m x 10^{e}, en machine l'exposant est modifié (avec un biais de 50) et tient dans un octet, la figure étant composée de 7 chiffres décimaux dont le premier est différent de 0.



On peut alors représenter tous les nombres dont la valeur absolue est comprise entre $0,1 \times 10^{-49}$ et $0,9999999 \times 10^{49}$.

Les opérations arithmétiques flottantes sont réalisées à l'aide de sous-programmes implantés à un endroit fixe en mémoire. Les temps d'exécution peuvent atteindre quelques centaines de millisecondes.

Il est évident que l'emploi systématique de ce genre d'instructions (en fait des macro-instructions) au cours de l'interprétation du programme BASIC, va augmenter les temps d'exécution.

CHAPITRE IV

ANALYSE SYNTAXIQUE

IV-1 - RAPPELS

Rappelons que le but d'une analyse syntaxique consiste à déterminer si oui ou non, les phrases du langage-cible sont bien formées, ce qui revient à vérifier qu'elles appartiennent ou non au langage défini sur la grammaire considérée.

IV-2 - EXPRESSIONS ARITHMETIQUES - INTRODUCTION AUX METHODES PRATIQUES

La notion d'expression arithmétique est familière au lecteur. On sait qu'une expression se compose d'un ensemble d'opérandes séparés par des signes d'opération, avec ou sans parenthèses.

Cela se traduit du point de vue grammatical par un ensemble V_T tel que :

$$V_T = \{ A, B, C, \ldots, Z, +, -, *, /, (,), \# \}$$

Remarquons qu'ici on suppose que les opérandes sont réduits aux variables littérales, qu'il n'y a pas de nombres. Nous avons introduit le caractère # , comme délimiteur d'achèvement d'expression, dans V_T.

Nous savons aussi qu'il existe une hiérarchie des opérations à effectuer, et par conséquent des signes d'opération. La multiplication, située au même niveau que la division, a priorité sur l'addition, qui est de même niveau que la soustraction.

Ainsi, A + B * C - D s'évalue comme A+(B * C) - D.

Quelle que soit la méthode d'analyse utilisée, il importe de bien évaluer une expression, plus précisément de préparer son interprétation (ou sa génération) correcte.

IV-2-1 - FORME POLONAISE

La mise sous forme polonaise (ainsi appelée car elle est due au mathématicien polonais £ukasiewicz) permet de s'affranchir des parenthèses et permet d'effectuer les opérations arithmétiques sans aucune ambiguïté. Nous ne citerons que la forme polonaise post-fixée, où les signes d'opération n'apparaissent qu'après les opérandes sur lesquels ils agissent.

A + (B * C) - D devient : A B C * + D -

En parcourant la chaîne sous forme polonaise et en s'arrêtant sur le premier signe d'opération rencontré, on trouve ABC* , il suffit de faire agir * sur B et C, pour faire B * C = P, on continue, soit AP+, qu'on effectue A + P = Q, il reste QD-, c'est-à-dire Q - D = R. On a affecté à R la valeur de l'expression.

Sous une autre forme, on peut écrire :

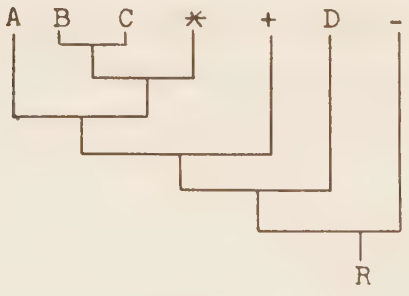


Figure IV-1

On conçoit qu'il soit facile d'automatiser ce genre de travail, sous réserve de disposer des formes polonaises. Ces dernières sont produites par un algorithme assez simple s'appuyant sur la grammaire d'expressions arithmétiques.

IV-2-2 - Méthode des piles de données (SAMELSON ET BAUER)

On utilise deux piles, l'une, V, pour les opérandes (ou variables) suivie par le pointeur PV, l'autre, OP, pour les symboles d'opération, suivie par le pointeur POP.

La méthode consiste à empiler les opérandes tant qu'on ne fait pas de réduction et à faire une réduction quand on reconnaît une opération valable. Pour tenir compte des priorités, on empile les signes d'opération jusqu'à ce que le dernier empilé ait une priorité supérieure à celle du signe courant.

Le test de priorité se fait par consultation d'une table à deux entrées, l'une des entrées parcourue en fonction du dernier caractère empilé dans OP, l'autre parcourue en fonction du caractère testé actuellement.

Une réduction se fait en réalisant l'opération :

$$((PV) - 2) \circ ((POP) - 1) \circ ((PV) - 1) \longrightarrow ((PV) - 2)$$

Exemple : A + B #

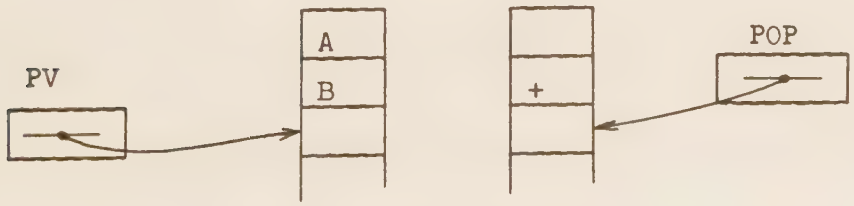


Figure IV-2

Remarquons qu'en analyse, il n'est pas nécessaire de réaliser véritablement l'opération. En fait, on fera comme si on l'effectuait, puisqu'il suffit d'avoir vérifié que la structure est bien formée.

Nous pouvons remarquer sur ces deux exemples, qu'en fait, analyses syntaxique et sémantique, sont liées. Du moins, il est très avantageux de pouvoir faire une analyse syntaxique permettant de passer facilement à la phase d'interprétation en incluant seulement des modules de traitement spéciaux. Cette façon d'aborder le problème a été une de nos préoccupations constantes.

Nous allons maintenant examiner les différentes méthodes d'analyse utilisables avec des grammaires développées du même type que la grammaire du BASIC.

IV-3 - METHODES D'ANALYSE

Il existe deux grandes classes de méthodes d'analyse :

- les méthodes d'analyse syntaxique descendante (top-down),

- les méthodes d'analyse syntaxique ascendante (bottom-up).

L'analyse descendante repose sur le principe suivant : Partant de l'axiome, on cherche à engendrer la chaîne terminale proposée. On procède de façon inverse en analyse ascendante, partant de la chaîne terminale proposée, on cherche à la réduire à l'axiome de la grammaire.

Nous allons prendre un exemple, pour préciser ces notions.

Soit une grammaire dont les éléments non-terminaux sont :

$$V_N = \{ \underset{\text{programme}}{<\text{PROG}>}, \underset{\text{expression}}{<\text{E}>}, \underset{\text{terme}}{<\text{T}>}, \underset{\text{facteur}}{<\text{F}>}, \underset{\text{variable}}{<\text{V}>} \}$$

$S = <\text{PROG}>$

et dont le vocabulaire terminal est $V_T = \{A, B, \ldots, Y, Z, +, *, (,), \#\}$

Ecrivons la grammaire en BNF :

$<\text{PROG}> ::= \#<\text{E}>\#$

$<\text{E}> ::= <\text{T}> \mid <\text{E}> + <\text{T}>$

$<\text{T}> ::= <\text{F}> \mid <\text{T}> * <\text{F}>$

$<\text{F}> ::= <\text{V}> \mid (<\text{E}>)$

$<\text{V}> ::= A \mid B \mid C \mid \ldots \mid Z$

Soit la chaîne à analyser :

A + B * C

L'arbre syntaxique lui correspondant est le suivant :

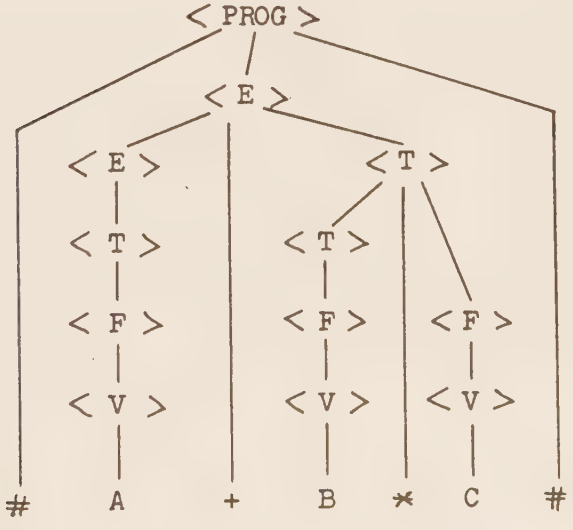
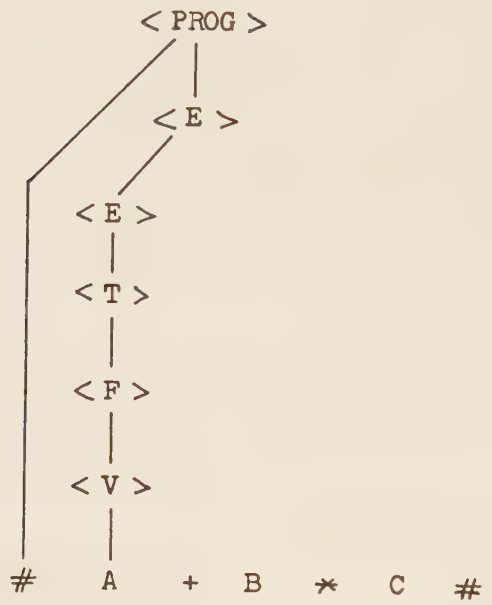


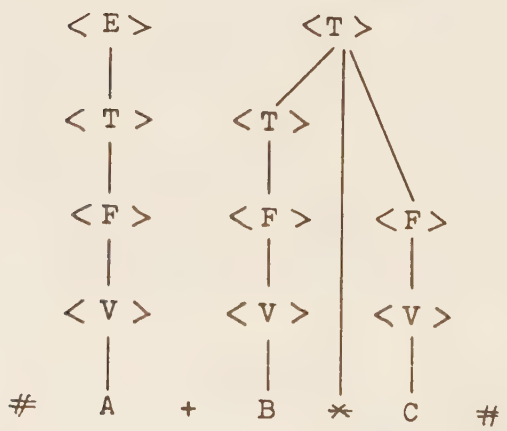
Figure IV-3

Cet arbre est obtenu en fin d'analyse. Nous allons voir son image en cours d'analyse selon la méthode utilisée. L'analyse se fait de gauche à droite, c'est le cas le plus général.



Arbre partiel en analyse descendante

Figure IV-4



Arbre partiel en analyse ascendante

Figure IV-5

Les différences sont essentielles, dans le premier cas, l'axiome est déjà en place, et on a descendu toute la structure jusqu'à l'élément terminal A, dans le second cas, on a remonté de A jusqu'à < E > , puis par manque d'information immédiate, on a analysé toute la structure B * C.

IV-4 - ANALYSE DESCENDANTE

Nous avons vu qu'en analyse descendante, on se propose de mécaniser le processus qui permet, par exploration de la grammaire en prenant l'axiome comme élément initial, de générer une chaîne terminale identique à celle qu'on analyse. C'est pourquoi, on dit aussi que cette analyse est prédictive.

Remarquons qu'en pratique, il n'y a pas de génération d'une chaîne à proprement parler, mais plutôt vérification de la présence dans la chaîne terminale, de l'élément qu'on s'attend à y trouver.

On sent bien, a priori, qu'il y a peu de chances de faire du premier coup les bonnes prédictions, ne serait-ce qu'à cause des alternatives possibles dans la grammaire. Ce problème se traduit en pratique par de multiples essais généralement partiels, qui obligent parfois à reprendre le processus d'analyse en amont et donc à reculer dans le texte d'entrée (phénomène de recul ou "backup").

Illustrons cet inconvénient par l'exemple suivant :

Soit la grammaire :

1 < AFFECT > ::= # < V > = < EA > #

2 < EA > ::= < T > | < T > { < SA > < T > }

3 < T > ::= < F > | < F > { < SM > < F > }

4 < F > ::= < P > | < P > { ↑ < P > }

5 < P > ::= < V > | (< EA >)

6 < V > ::= A | B | C | ... | Z

7 < SA > ::= + | -

8 < SM > ::= * | /

Précisons tout de suite la signification des accolades : elles traduisent le fait qu'on peut avoir une suite d'occurences (infinie a priori), c'est un moyen élégant de traduire les structures répétitives du type A + B - C + D + E.

Soit la chaîne terminale : A = (B + C / D) ↑ E.
(↑ est le symbole d'exponentiation utilisé en BASIC par exemple).

L'arbre partiel auquel conduit l'application des règles en prenant leurs premiers alternants est le suivant :

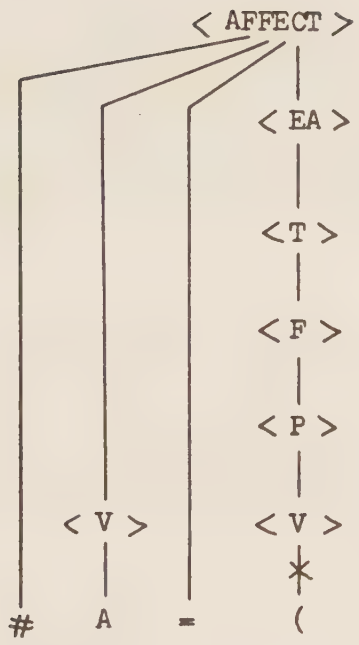


Figure IV-6

On a prédit la présence d'une variable, ce qui se révèle faux. Il faut donc remonter dans la structure. Il suffit ici, pour avancer correctement, de prendre le second alternant dans la règle 5 , et l'arbre partiel est alors modifié de la façon suivante :

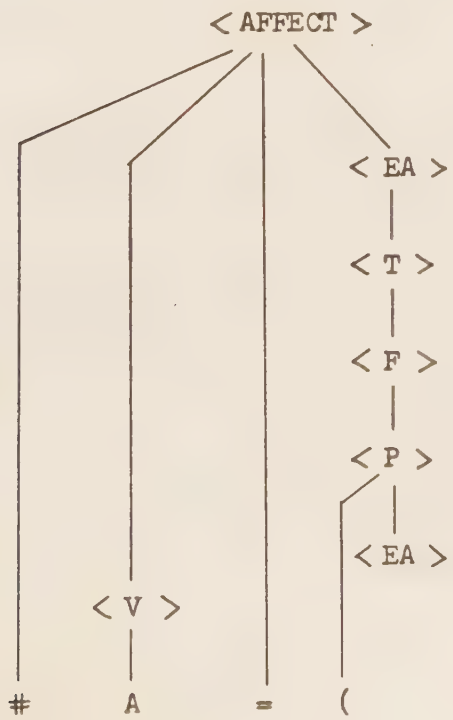


Figure IV-7

Nous avons supposé ici, que nous reprenons seulement la règle 5 , mais nous pouvons concevoir facilement que dans un cas plus complexe, avec une grammaire plus développée, nous sommes amenés à supprimer toute une partie de l'analyse avant de faire une nouvelle tentative, ce qui suppose une mémorisation d'informations suffisantes pour permettre le recul.

Les multiples tentatives du processus d'analyse descendante ont le gros inconvénient d'allonger considérablement les temps d'analyse, ce qui oblige à rejeter cette méthode dans le cas d'une application sur une petite machine dont la durée des instructions n'est pas négligeable.

Remarquons qu'il faut écrire d'une autre façon, les règles récursives à gauche, sans quoi le processus d'analyse descendante entre dans une boucle sans fin, en recherchant indéfiniment le même élément, e.g la règle suivante :

$$< \text{ID VARIABLE} > ::= < \text{ID VARIABLE} > \left\{ < \text{LETTRE} > \mid < \text{CHIFFRE} > \right\}$$

IV-5 - ANALYSE ASCENDANTE

IV-5-1 - Réductions

Nous savons que la méthode ascendante consiste à partir de la chaîne terminale et à la réduire progressivement jusqu'à l'axiome.

On a souvent intérêt dans le cadre de cette méthode à présenter la grammaire sous forme de réductions : c'est l'écriture de la grammaire à l'envers.

Par exemple, si l'on reprend la dernière grammaire que nous avons utilisée, en convenant de prendre le symbole $\longrightarrow$ pour représenter une réduction, on obtient :

$$\# \; \langle V \rangle = \langle EA \rangle \; \# \longrightarrow \langle AFFECT \rangle$$

$$\langle T \rangle \mid \langle T \rangle \{ \langle SA \rangle \langle T \rangle \} \longrightarrow \langle EA \rangle$$

$$\langle F \rangle \mid \langle F \rangle \{ \langle SM \rangle \langle F \rangle \} \longrightarrow \langle T \rangle$$

$$\langle P \rangle \mid \langle P \rangle \{ \uparrow \langle P \rangle \} \longrightarrow \langle F \rangle$$

$$\langle V \rangle \mid (\langle EA \rangle) \longrightarrow \langle P \rangle$$

$$A \mid B \mid C \mid \ldots Y \mid Z \longrightarrow \langle V \rangle$$

$$+ \mid - \longrightarrow \langle SA \rangle$$

$$* \mid / \longrightarrow \langle SM \rangle$$

On effectuera une réduction si celle-ci est possible et seulement si elle ne peut être remise en question dans la suite de l'analyse. Cette dernière condition est importante puisqu'elle détermine en fait le "non-recul" de l'analyse et assure la supériorité de la méthode relativement au temps d'exécution, tout au moins.

IV-5-2 - Méthode de EICKEL, PAUL, BAUER et SAMELSON

Nous avons décrit au § IV-2-2, la méthode des piles de données qui reste une méthode pratique. Elle repose néanmoins sur le même type de considérations que celles exposées ici.

Soit un langage L, dont les réductions sont du type :

$$RS \longrightarrow T \qquad \text{ou} \qquad R \longrightarrow T$$

Il n'y a qu'un ou deux éléments dans la partie gauche de chaque réduction. On peut toujours transformer une grammaire à indépendance contextuelle de façon à n'avoir que des réductions de ce type. Il suffit d'augmenter le nombre de réductions, et de se donner des éléments supplémentaires dans V_N. Par exemple :

$X_1\ X_2\ X_3\ X_4 \longrightarrow U$ se transforme en

$X_1\ X_2 \longrightarrow V_1$

$V_1\ X_3 \longrightarrow V_2$

et $V_2\ X_4 \longrightarrow U$

Il faut alors ajouter V_1 et V_2 à $\{V_N\}$.

Soit r une réduction, et $\mathcal{R}$ l'ensemble des réductions.

Soit une table à 3 dimensions, construite à partir de la grammaire, contenant un doublet nN pour chaque triplet de symboles (RST).

Soit une pile dont les éléments sont R et S, T étant le symbole suivant dans la chaîne d'entrée.

Alors dans l'état n, on exécute l'action N.

n = 1 Si $\exists\ r \in \mathcal{R}$ telle que r : $RS \longrightarrow U$

N = réduire RS en U dans la pile.

n = 2 Si $\exists\ r \in \mathcal{R}$ telle que r : $S \longrightarrow U$

N = réduire S en U dans la pile

n = 3 $\nexists\ r \in \mathcal{R}$ telle que r : $RS \longrightarrow U$ ou $S \longrightarrow U$

N = ajouter T dans la pile, prendre le symbole suivant.

On voit que s'il y a beaucoup de réductions dans la grammaire (ou beaucoup d'éléments dans V_N), la table est très volumineuse, d'autre part, le temps de recherche dans la table peut être important. Ces deux inconvénients nous font rejeter cette méthode.

IV-5-3 - Définitions

Soit une grammaire G $\quad G = (V_T, V_N, S, P)$

On appelle forme sententielle une chaîne x obtenue par dérivation de S telle que : $S \overset{*}{\Longrightarrow} x$ (l'ensemble des formes sententielles est plus vaste que le langage).

On appelle phrase u d'une forme sententielle w relativement au non-terminal U, la chaîne u telle que :

α) $\quad w = x\,u\,y$ soit forme sententielle

β) $\quad S \overset{*}{\Longrightarrow} x\,U\,y$ et $U \overset{*}{\Longrightarrow} u$

La phrase est simple si la dérivation U à u est simple.

$$U \Longrightarrow u$$

Pour une chaîne terminale donnée, on appelle gerbe la phrase simple la plus à gauche.

De façon plus pratique, une gerbe est l'ensemble de feuilles adjacentes issues du même noeud, le plus à gauche dans l'arbre d'analyse syntaxique. On étend la notion de gerbe à toutes les parties du graphe obtenues par réduction, en généralisant la notion de feuille, à tout noeud obtenu par "filiation".

Exemple : Soit la grammaire :

< ORDRE > ::= < ordre READ > | < ordre INPUT >

< ordre READ > ::= < READ > < Variable >

< ordre INPUT > ::= < INPUT > < Variable >

< READ > ::= READ

< INPUT > ::= INPUT

< Variable > ::= A | B | C | .... | Z

Soit la chaîne READ K

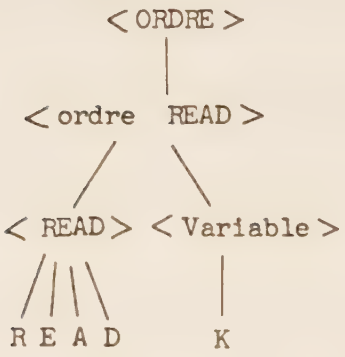


Figure IV-8

La première gerbe est ici READ.

En analyse ascendante, le problème est la recherche à tout instant, de la phrase simple la plus à gauche.

IV-5-4 - Grammaires de simple précédence - Méthode de WIRTH et WEBER

On se propose d'essayer d'établir une relation entre deux symboles appartenant au vocabulaire total.

Soient S_i et $S_j \in V$, supposons qu'ils font partie d'une forme sententielle. Si ces deux symboles peuvent entrer dans un ensemble réductible, nous dirons qu'ils appartiennent à une gerbe.

Il n'existe que 3 types de relation entre S_i et S_j.

IV-5-4-1 - Relation $S_i \lessdot S_j$

S_j fait partie d'une gerbe, mais pas S_i, alors on note $S_i \lessdot S_j$. (Il faut alors qu'il existe une réduction du type $S_j xxx \longrightarrow x$).

$$S_i \underbrace{S_j xxx}_{\text{gerbe}}$$

Figure IV-9

IV-5-4-2 - Relation $S_i \doteq S_j$

S_i et S_j font partie de la même gerbe.
($\exists$ réduction de la forme $xxx\ S_i\ S_j xxx \longrightarrow x$)

$$\underbrace{xxx\ S_i\ S_j xxx}_{\text{gerbe}}$$

Figure IV-10

IV-5-4-3 - Relation $S_i \geq S_j$

S_i seul fait partie d'une gerbe, S_j étant un symbole terminal ($\exists$ réduction de la forme $xx\ S_i \longrightarrow x$ et $S_j \in V_T$)

$$\underbrace{xxx\ S_i}_{\text{gerbe}}\ S_j xx$$

Figure IV-11

Remarque :

S'il n'existe pas de forme sententielle englobant la séquence $xxx\ S_i\ S_j\ xxx$, il ne peut pas y avoir de relation entre S_i et S_j.

IV-5-4-4 - Simple précédence

Une grammaire est dite de simple précédence si pour toute paire de symboles, il existe au plus une relation de précédence.

IV-5-4-5 - Moyens d'obtention des relations de précédence

On dit que l'on a la relation U COMMENCE PAR v si la grammaire contient une règle de la forme $U \Longrightarrow vxxx$.

Si $U \overset{*}{\Longrightarrow} vxxx$, alors on a : U COMMENCE PAR* v.

De même, on a U FINIT PAR v, s'il existe $U \Longrightarrow xxv$.

Appliquons ces définitions et réécrivons les relations de précédence entre tous les éléments de V pris deux à deux.

IV-5-4-5-1 - $S_i \doteq S_j$

Règle inchangée, il suffit qu'il existe une production de la forme :

$$U \Longrightarrow xxx\ S_i\ S_j\ xx$$

IV-5-4-5-2 - $S_i \lessdot S_j$

Il faut qu'il existe une production de la forme :

$$U \Longrightarrow xxx\ S_i\ S_k\ xxx$$

avec S_k COMMENCE PAR* S_j

IV-5-4-5-3 - $S_i \gtrdot S_j$

Il faut et il suffit qu'il existe une production de la forme :

$$U \Longrightarrow xxx\ S_k\ S_l\ xxx$$

où S_k FINIT PAR* S_i et S_l COMMENCE PAR* S_j

($S_j \in V_T$)

IV-5-4-6 - Application

Soit la grammaire :

1 $<S> ::= <E>$

2 $<E> ::= <E1>$

3 $<E1> ::= <T> \mid <E1> + <T>$

4 $<T> ::= <T1>$

5 $<T1> ::= <F> \mid <T1> * <F>$

6 $<F> ::= I \mid (<E>)$

Nous pouvons représenter dans une matrice les relations existant entre les symboles, c'est la matrice de précédence.

Etablissons les relations COMMENCE PAR* et FINIT PAR*

	COMMENCE PAR*	FINIT PAR*
<S>	<E>, <E1>, <T>, <T1>, <F> , I, (	<E>, <E1>, <T>, <T1>, <F> , I,)
<E>	<E1>, <T>, <T1>, <F>, I, (	<E1>, <T>, <T1>, <F> , I,)
<E1>	<T> , <T1>, <F> , I,(	<T> , <T1> , <F> , I,)
<T>	<T1> , <F >, I,(	<T1> , <F >, I,)
<T1>	<F> , I,(	<F >, I,)
<F>	I,(	I,)

Figure IV-9

Les productions 3, 5 et 6 nous donnent directement les relations $\doteq$:

$$<E1> \doteq + \doteq <T>$$
$$<T1> \doteq * \doteq <F>$$
$$(\doteq <E> \doteq)$$

Ce sont les seules productions qui nous permettent de déterminer les relations $<$ et $>$:

$<E1>$ + $<T>$

S_i S_k COMMENCE PAR* S_j pour $\lessdot$

$<T1>$ * $<F>$

S_i S_k COMMENCE PAR* S_j pour $\lessdot$

($<E>$)

S_i S_k COMMENCE PAR* S_j pour $\lessdot$

$<E1>$ + $<T>$

S_k $S_1 = S_j$ pour $\gtrdot$

FINIT PAR* S_i

$<T1>$ * $<F>$

S_k $S_1 = S_j$ pour $\gtrdot$

FINIT PAR* S_i

($<E>$)

S_k $S_1 = S_j$ pour $\gtrdot$

FINIT PAR* S_i

On regroupe tous ces résultats dans la matrice de précédence, figure IV-10.

S_i \ S_j	<S>	<E>	<E1>	<T>	<T1>	<F>	I	(	)	*	+
<S>											
<E>									$\doteq$		
<E1>									$>$		$\doteq$
<T>									$>$		$>$
<T1>									$>$	$\doteq$	$>$
<F>									$>$	$>$	$>$
I									$>$	$>$	$>$
(		$\doteq$	$<$	$<$	$<$	$<$	$<$	$<$			
)									$>$	$>$	$>$
*						$\doteq$	$<$	$<$			
+				$\doteq$	$<$	$<$	$<$	$<$			

Figure IV-10

On voit qu'on peut regrouper les 4 solutions, $\doteq$,$<$, $>$, et $\not\exists$ relation, à l'aide de 2 digits binaires, la taille de la matrice de précédence est alors égale à $2\ N^2$ bits, si N est le nombre de symboles du vocabulaire total. Sachant qu'un langage de programmation demande au moins une centaine de symboles, il faut réserver 20 000 bits, soit 2 500 octets. C'est beaucoup trop pour la capacité de notre mémoire centrale.

IV-5-4-7 - Fonctions de précédence

Dans certains cas, on peut gagner de la place en représentant les relations de précédence à l'aide de deux fonctions F et G telles que :

$$S_i \lessdot S_j \Longrightarrow F(S_i) < G(S_j)$$

$$S_i \doteq S_j \Longrightarrow F(S_i) = G(S_j)$$

$$S_i \gtrdot S_j \Longrightarrow F(S_i) > G(S_j)$$

Si ces fonctions existent, on peut les regrouper dans une table de 2 x N éléments. Si l'on ne dispose que d'une machine ne permettant pas l'extraction facile et rapide de bits dans un octet, il faut prendre un octet pour stocker la valeur de F ou G, soit 2 x N octets au total, ce qui n'est pas prohibitif.

Par contre, si l'on a gagné de la place, on perd du temps en analyse, car il faut effectuer davantage d'étapes d'analyse pour finalement s'apercevoir que la phase analysée n'appartient pas au langage. Ceci provient du fait que si l'on trouve 2 symboles entre lesquels il n'y a pas de relation de précédence, on peut affirmer qu'il y a une erreur. Cette possibilité n'existe pas avec les fonctions. Cet inconvénient et les ambiguïtés qui subsistent sur les fonctions F et G, nous font écarter cette méthode.

IV-5-5 - Grammaires à précédence d'opérateurs - Méthodes de FLOYD

Si la grammaire est telle qu'il n'existe pas de relation de la forme $U \longrightarrow \alpha AB \beta$ avec $A, B \in V_N$, la grammaire est dite d'opérateurs.

Soient U, A, B $\in$ V_N ; t_1, $t_2 \in V_T$. On définit les relations de précédence de la façon suivante :

1°/ - $t_1 \doteq t_2$ s'il existe $U \longrightarrow \alpha\, t_1\, t_2\, \beta$

ou s'il existe $U \longrightarrow \alpha\, t_1\, A\, t_2 \beta$

2°/ - $t_1 \lessdot t_2$ s'il existe $U \longrightarrow \alpha\, t_1\, A\, \beta$

avec $A \overset{*}{\Longrightarrow} t_2 \gamma$ ou $A \overset{*}{\Longrightarrow} B\, t_2\, \gamma$

3°/ - $t_1 \gtrdot t_2$ s'il existe $U \longrightarrow \alpha\, A\, t_2\, \beta$

avec $A \overset{*}{\Longrightarrow} \gamma\, t_1$ ou $A \overset{*}{\Longrightarrow} \gamma\, t_1\, B$

Si, pour chaque paire de symboles terminaux, on détermine au plus une de ces relations, on dit que l'on a une grammaire à précédence d'opérateurs.

Remarquons que la grammaire utilisée au § IV-5-4-6 en est une.

Les relations de précédence ne permettent pas ici l'analyse du langage, une gerbe ne pouvant pas être constituée d'un seul élément non-terminal, puisque ce dernier n'est pas pris en compte par la matrice de précédence, d'où l'introduction d'une nouvelle notion.

Une <u>phrase première</u> (ou primitive) est une phrase contenant au moins un symbole terminal, mais qui ne contient pas d'autre phrase première. La phrase première la plus à gauche est la sous-chaîne $X_j\, A_j \ldots X_i\, A_i\, X_{i+1}$ la plus à gauche telle que :

$$A_{j-1} \lessdot A_j \doteq A_{j+1} \ldots A_{i-1} \doteq A_i \gtrdot A_{i+1}$$

On réduira de gauche à droite, les phrases premières successivement les plus à gauche.

La méthode d'obtention des relations de précédence est du même genre que celle utilisée pour les grammaires de simple précédence. Dans la grammaire du BASIC, il existe des productions de la forme $U \longrightarrow \alpha AB\beta$ où $A, B \in V_N$

$\alpha, \beta \in V^*$ (peuvent être vides)

le BASIC n'a donc pas de grammaire à précédence d'opérateurs.

Nous n'insisterons donc pas sur cette méthode. Concluons en disant que c'est la méthode la plus pratique pour analyser les langages où il ne rentre que des opérandes avec les symboles terminaux qui les relient, typiquement : les langages d'expressions arithmétiques.

IV-5-6 - Grammaires de précédence non simple

Il existe des grammaires de précédence où il faut utiliser plusieurs symboles pour déterminer une gerbe et non plus deux comme avant.

Nous dirons simplement que cette méthode requiert davantage de matrices de tailles plus élevées qu'en simple précédence, cet inconvénient étant suffisant pour faire abandonner l'étude.

IV-5-7 - Analyse ascendante contextuelle

Nous avons vu qu'il est bien préférable de ne jamais revenir en arrière lors d'une analyse. On peut, pour remplir cette condition de "non-recul", être amené à tester le contexte.

Les différents symboles d'opération et signes de ponctuation nous serviront de contexte postérieur pour déclencher les prises de décision. Le contexte antérieur plus ou moins limité interviendra, particulièrement, pour lever les ambiguïtés qui peuvent exister sur la signification à donner à un symbole terminal (par exemple, savoir si $\times$ est le signe de multiplication scalaire, ou le signe de produit matriciel).

Remarque importante :

A partir de maintenant, nous appellerons opérateur, le groupe opérande-symbole qui agit sur l'opérande suivant. On verra plus tard que cette méthode favorise l'insertion des modules d'interprétation.

Nous choisissons des délimiteurs spéciaux d (début) et f (fin) pour terminer correctement l'analyse.

Soit un langage d'expressions arithmétiques avec une instruction d'affectation, la séparation entre les diverses affectations se faisant par l'emploi de ";".

Soit $\mathcal{R}$, l'ensemble des réductions suivantes :

1 A B C ... Z $\longrightarrow$ <Variable>

2 <Variable> = $\longrightarrow$ <partie gauche>

3 ($\longrightarrow$ <ouverture parenthèse>

4 <ouverture parenthèse><expr. arithmétique>$\longrightarrow$<(exp. arith.>

5 <Variable> <(expr. arith.>) $\longrightarrow$ <primaire>

6 <facteur> ↑ ⟶ <opérateur exponentiation>

7 <primaire> | <op. exp.> <primaire> ⟶ <facteur>

8 + | - ⟶ <signe addition>

9 * | / ⟶ <signe multiplication>

10 <exp. arith. S> <signe add.> ⟶ <opérateur addition>

11 <terme> <signe mult.> ⟶ <opérateur multiplication>

12 <facteur> | <op. mult.> <facteur> ⟶ <terme>

13 <terme> | <op. add.> <terme> ⟶ <exp. arith. S>

14 <exp. arith. S> ⟶ <expression arithmétique>

15 <partie gauche> <exp. arith.> ⟶ <instruction>

16 <instruction> | <suite d'inst.;> <inst.> ⟶ <suite d'inst.>

17 <suite d'instruction> ; ⟶ <suite d'inst. ;>

18 d <suite d'instructions> f ⟶ <programme>

IV-5-7-1 - Graphe de la grammaire

On indique la concaténation par le symbole ⊙ .
Les flèches pointant vers ce symbole indiquent les éléments qui sont concaténés ainsi que leur ordre de concaténation.

Remarquons qu'ici, il n'y a pas de réduction conditionnelle, il n'y aucune ambiguïté sur les symboles, ni sur les éléments non-terminaux (c'est normal puisqu'en fait, on est dans le contexte des expressions arithmétiques).

Il est moins fastidieux de remplacer les éléments syntaxiques par des numéros, puis de ne travailler qu'avec ceux-ci. Nous numérotons l'axiome par 1, l'ordre suivant est arbitraire.

< 1 >	< programme >
< 2 >	< suite d'instructions >
< 3 >	< instruction >
< 4 >	< partie gauche >
< 5 >	< opérateur exponentiation >
< 6 >	< opérateur multiplication >
< 7 >	< opérateur addition >
< 8 >	< expression arithmétique >
< 9 >	< expression arithmétique S >
< 10 >	< terme >
< 11 >	< facteur >
< 12 >	< primaire >
< 13 >	< variable >
< 14 >	< signe multiplication >
< 15 >	< signe addition >
< 16 >	< suite d'instructions ; >
< 17 >	< (expression arithmétique >
< 18 >	< ouverture parenthèse >

On peut alors recopier les règles de grammaire à l'aide de ces numéros.

1 A | B | C | ... | Y | Z ⟶ <13>

2 <13> = ⟶ <4>

3 (⟶ <18>

4 <18><8> ⟶ <17>

5 <13> | <17>) ⟶ <12>

6 <11> ↑ ⟶ <5>

7 <12> | <5><12> ⟶ <11>

8 + | - ⟶ <15>

9 * | / ⟶ <14>

10 <9> <15> ⟶ <7>

11 <10> <14> ⟶ <6>

12 <11> | <6><11> ⟶ <10>

13 <10> | <7><10> ⟶ <9>

14 <9> ⟶ <8>

15 <4> <8> ⟶ <3>

16 <3> | <16><3> ⟶ <2>

17 <2> ; ⟶ <16>

18 d <2> f ⟶ <1>

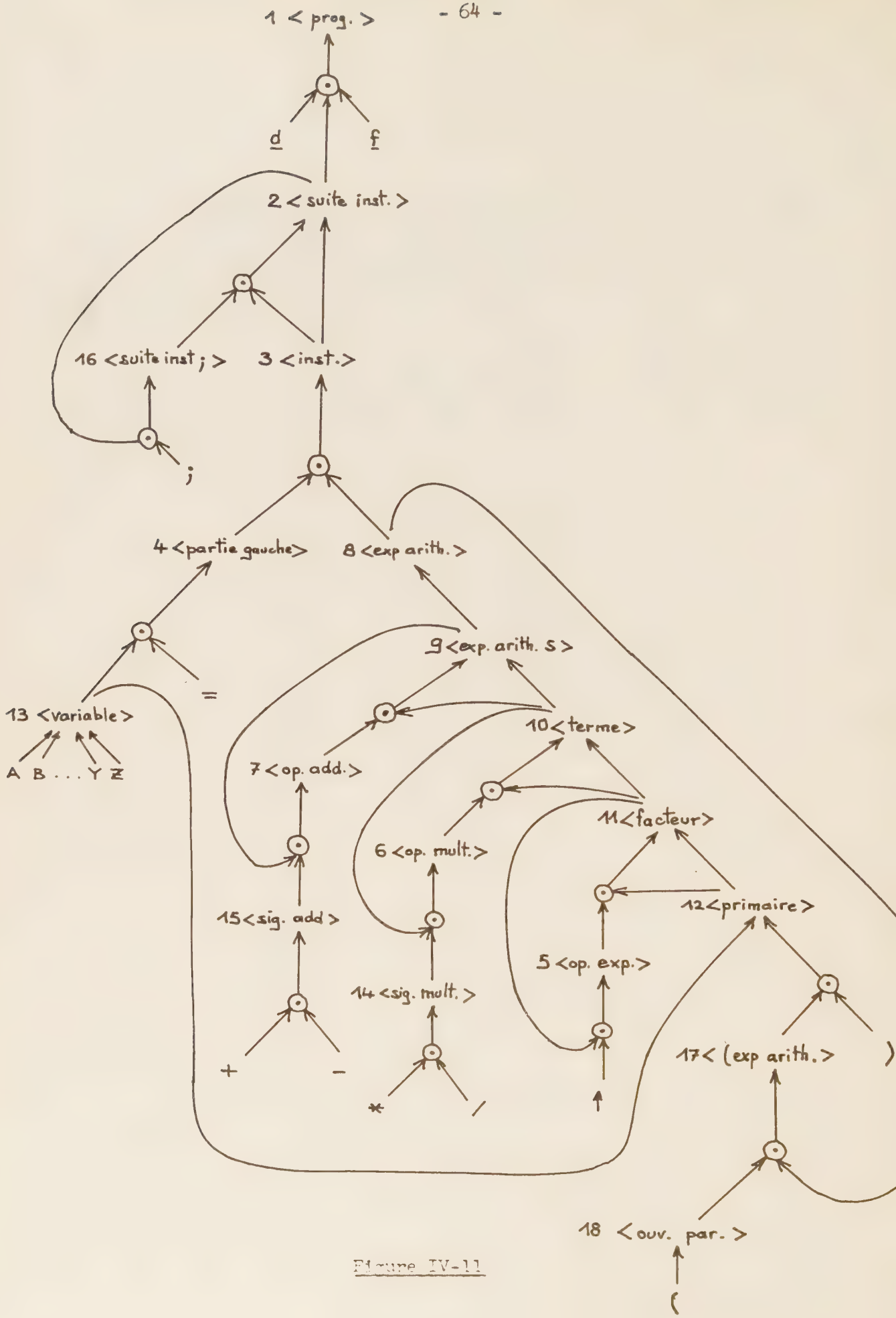
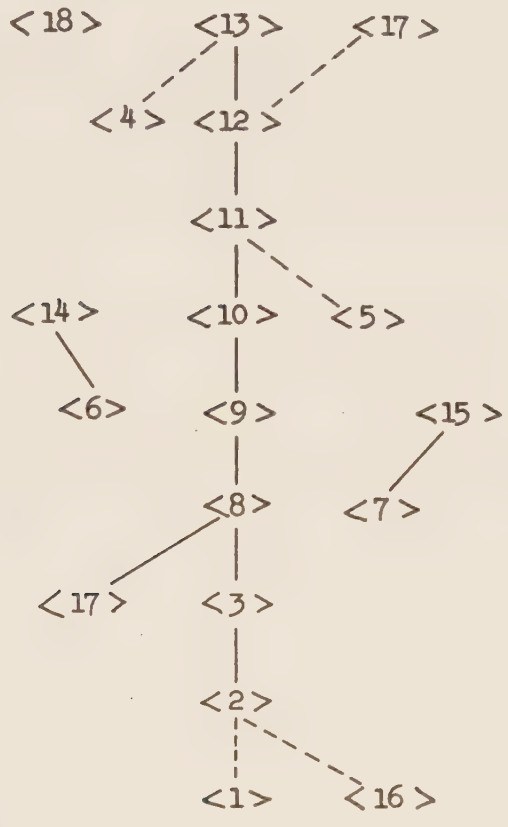

1 < prog. >
d
f
2 < suite inst. >
16 < suite inst ; >
3 < inst. >
;
4 < partie gauche >
8 < exp arith. >
=
9 < exp. arith. S >
13 < variable >
A B . . . Y Z
10 < terme >
7 < op. add. >
11 < facteur >
6 < op. mult. >
12 < primaire >
15 < sig. add >
5 < op. exp. >
14 < sig. mult. >
+
-
*
/
17 < (exp arith. >)
18 < ouv. par. >
(

Figure IV-11

IV-5-7-2 - Graphe d'ordre partiel

On peut définir le graphe d'ordre partiel de la grammaire, figure IV-12, en considérant l'élément syntaxique écrit à droite dans une réduction, inférieur en ordre à l'élément syntaxique final du membre de gauche.



Graphe d'ordre partiel

Figure IV-12

Ce graphe est d'une importance vitale, car c'est lui qui nous indique dans quel ordre il faut faire l'analyse. En fait, la stratégie d'analyse apparaît clairement ; il faut commencer par l'élément <13> et descendre dans le graphe jusqu'à <1> , on voit qu'il peut y avoir une structure emboîtée (se traduisant au niveau de l'expression à analyser par une paire de parenthèses), ce qui oblige de <8> à <17> puis à reprendre <17>en haut, pour aller en <12>.

Remarquons qu'à part les entrées "éléments terminaux", il n'y a qu'une seule entrée, c'est <13> qui symbolise <variable> . Nous verrons au chapitre suivant qu'avec une grammaire plus évoluée, nous trouverons plusieurs entrées.

IV-5-7-3 - Matrice de sélection

Sachant que les éléments permettant d'effectuer les regroupements sont des symboles terminaux se comportant en séparateurs, il est intéressant de savoir quels sont les éléments syntaxiques qui peuvent précéder le caractère en analyse.

La méthode est analogue à l'obtention des relations de précédence. Ici, on utilise les matrices booléennes.

Soit la matrice [A] , qui donne, par lecture directe des règles de réduction, les éléments syntaxiques pouvant précéder les séparateurs.

[A]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
f		1																
+ | -									1									
× | /										1								
↑											1							
)																	1	
=													1					
;		1																

Figure IV-13

On n'a fait apparaître dans [A] que les éléments terminaux pouvant jouer le rôle de contexte postérieur, c'est pour cela que "(" n'y figure pas, ni "d" d'ailleurs.

On sait que la grammaire donne les règles de bonne formation du langage, on peut appliquer ces règles un nombre de fois quelconque. Il est donc intéressant de savoir quel élément syntaxique peut précéder un séparateur après application des règles un nombre quelconque de fois. Il faut alors chercher la fermeture transitive d'une matrice [M] obtenue à partir des règles, en indiquant pour chaque élément obtenu par réduction quel est l'élément final du membre de gauche permettant cette réduction.

[M] =

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1																		
2			1															
3								1										
4																		
5																		
6														1				
7															1			
8									1									
9										1								
10											1							
11												1						
12													1					
13																		
14																		
15																		
16																		
17								1										
18																		

Figure IV-14

$$[M^*] = [M] \vee ([M] \cdot [M]) \vee ([M] \cdot [M] \cdot [M]) \ldots$$

La matrice $[S]$, permettant de savoir pour chaque séparateur, quel élément syntaxique peut le précéder, est obtenue par la relation :

$$[S] = \left\{[I] \vee [M^*]\right\} . [A]$$

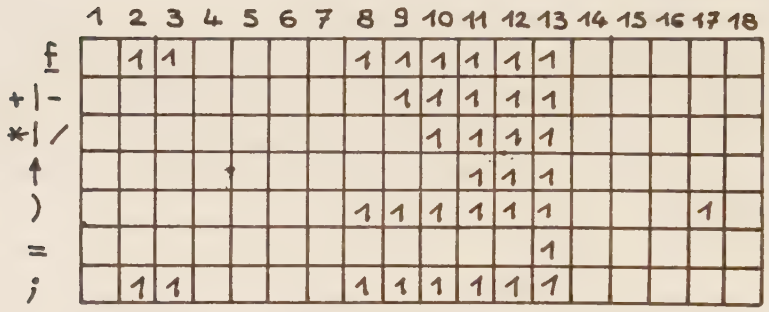


Figure IV-15

On voit que le signe d'affectation (signe égal) ne peut être précédé que par <13> (<variable>), alors que ↑ peut suivre une <variable> , un <primaire>, un <facteur>.

On peut concevoir un algorithme d'analyse exploitant les résultats de cette table, néanmoins, la matrice à explorer atteint une taille importante pour une grammaire développée, par exemple en BASIC la matrice M atteint $136^2 \simeq 18\,500$ éléments et S, 136 x 16 = 2176 éléments, soit dans tous les cas, un encombrement prohibitif ou un temps d'extraction des bits trop long.

La solution qui s'impose revient à utiliser implicitement les résultats de cette matrice. On peut, par exemple, opérer la réduction <13> à <12>, puis <12> à <11> et alors tester si le caractère en analyse est le symbole d'exponentiation. Dans ce dernier cas, on effectue le regroupement <11>↑→<5>, ainsi on a bien utilisé les trois "1" qui figurent sur la ligne de "↑".

IV-5-7-4 - Arbre d'analyse syntaxique

Sachant qu'on effectue une analyse sans recul, tous les regroupements opérés deviennent définitifs. Si l'on garde en mémoire les éléments syntaxiques manipulés dans les regroupements, ainsi que l'ordre dans lequel ils apparaissent, on obtient les renseignements nécessaires à la construction de l'arbre d'analyse syntaxique, sous réserve de prévoir la mise en mémoire du nombre d'éléments regroupés, ce qui donne le nombre de branches pendantes pour chaque noeud.

IV-5-7-5 - Adaptation à l'interprétation

Cette méthode a l'énorme avantage de permettre des inclusions sémantiques sous la forme de modules s'intercalant dans certaines phases de l'analyse. Nous aurons loisir de développer plus avant ces points, puisque nous choisissons cette méthode.

CHAPITRE V

ANALYSEUR SYNTAXIQUE DU BASIC

V-1 - DEFINITION DE LA GRAMMAIRE BASIC

V-1-1 - Règles de production

<lettre> ::= 'A|B|C|...|Y|Z'
<chiffre> ::= '1|2|... 8|9|0'
<variable simple>::=<lettre>|<lettre><chiffre>

(1) <identificateur de tableau> ::=<lettre>
< guillemet > ::= '"'
(2) <caractère> ::= 'A|B|...|Z|0|1|...|9|+|-| ...'
<chaîne de caractères> ::=<caractère>|<chaîne de caractères><caractère>
<"chaîne de caractères"> ::=<"chaîne de caractères><guillemet>
<point> ::= '.'
<entier> ::=<chiffre><entier><chiffre>
<entier pointé>::=<entier><point>
<nombre décimal>::=<entier>|<entier pointé><entier>|<entier pointé>|
<point><entier>
<exposant>::= 'E'
<signe addition> ::= '+| -'
(3) <addition matricielle> ::= '+| -'
<opérateur exposant> ::=<exposant><signe addition>
<partie exposant> ::= <exposant><entier>|<opérateur exposant><entier>
<nombre> ::= <nombre décimal>|<nombre décimal><partie exponsant>
(4) <constante> ::= <nombre> |<signe addition><nombre>
<ouverture de parenthèse> ::= '('
<"chaîne de caractères >::= < guillemet ><chaîne de caractères>

(14) <(variable simple>::= <ouverture de parenthèse><variable simple>
<(expression> ::= <ouverture de parenthèse><expression>
<(expression)>::= <(expression>')'
<fonction prédéfinie>::= 'SIN|CØS|ABS|ATN|EXP|LØG|SQR|TAH'
<définition>::= 'FN'
<identificateur de fonction>::= <définition><lettre>
<virgule> ::= ','
<fonction 1>::=<identificateur de fonction><(variable simple>')'
<fonction 2>::=<identificateur de fonction><(expression)>|
<fonction prédéfinie><(expression)>
<primaire> ::=<variable>|<nombre>|<fonction 2>|(expression)>
(5) <liste de variables> ::=<variable>|<liste de variables><virgule>
<variable>
<opérateur d'exponentiation>::=<facteur> '↑'
<facteur> ::=<primaire>|<opérateur d'exponentiation><primaire>
<signe de multiplication>::= '* | /'
(3) <multiplication matricielle>::= '*'
<opérateur d'addition> ::=<somme S> <signe d'addition>
<opérateur de multiplication>::=<terme><signe de multiplication>
<terme> ::=<signe d'addition><facteur>|<opérateur de multipli-
cation><facteur>|<facteur>
<somme S>::= <terme>|<opérateur d'addition><terme>
<somme> ::= <somme S>
<mineur - majeur> ::= 'MIN | MAX'
<Min Max> ::= <minmax><mineur - majeur>
<minmax> ::= <somme>|<MinMax><somme>
(13) <signe de relation> ::= '<|<=|=|<>|>=|>'
(7) <signe d'affectation matricielle>::= '='
(8) <signe égal>::= '='
<partie gauche 1>::=<minmax><signe de relation>
<relation> ::=<partie gauche 1><minmax>
<expression>::=<minmax>
<lettre (>::= <lettre><ouverture de parenthèse>
<liste d'indices,> ::=<expression><virgule>

```
<liste d'indices> ::=<liste d'indices,><expression>|<expression>
<variable indicée>::=<lettre (><liste d'indices>')'
<variable> ::= <variable simple>|<variable indicée>
<partie gauche 2>::= <variable><signe égal>|
                     <partie gauche 2><variable><signe égal>
(9) <numéro de ligne>::= <entier>
(12) <partie gauche 3>::= <variable simple><signe égal>
<fonction 1 = >  ::= <fonction 1><signe égal>
<fonction définie> ::= <fonction 1 =><expression>
(10) <quantité> ::= <entier>
<quantité,> ::= <quantité><virgule>
<quantité 1>::= <quantité,><quantité>
<identificateur de tableau (> ::= <identificateur de tableau>
                                  <ouverture parenthèse>
<spécifications dimensions> ::= <identificateur de tableau (>
                                <quantité 1>')'|
                     <identificateur de tableau (><quantité>')'
(11) <dimension,> ::= <expression><virgule>
(11) <dimensions> ::= <dimension,><expression>|<expression>
<tableau> ::= <identificateur de tableau (><dimensions>')'|
              <identificateur de tableau>
<fonction initialisatrice> ::= 'ZER|CØN|IDN'
<fonction matricielle> ::= 'INV|TRN'
<fonction matricielle (>::=<fonction matricielle><ouverture de
                                                  parenthèse>
<fonction initialisatrice (>::=<fonction initialisatrice>
                               <ouverture de parenthèse>
<initialisation> ::=<fonction initialisatrice (><dimensions>')'
<opérateur matriciel>::=<(expression)><multiplication matricielle>|
              <identificateur de tableau><addition matricielle>|
              <identificateur de tableau><multiplication matric.>
<terme matriciel> ::=<opérateur matriciel><identificateur de
                                                  tableau>|
              <fonction matricielle (><identificat. de tableau>')'
```

```
<LET> ::= 'LET'
<ordre LET> ::= <LET> <partie gauche 2> <expression>
<IF> ::= 'IF'
<THEN> ::= 'THEN'
<expression IF> ::= <IF> <relation>
<THEN numéro de ligne> ::= <THEN> <numéro de ligne>
<ordre IF> ::= <expression IF> <THEN numéro de ligne>
<GØTØ> ::= 'GØTØ'
<ordre GØTØ> ::= <GØTØ> <numéro de ligne>
<GØSUB> ::= 'GØSUB'
<ordre GØSUB> ::= <GØSUB> <numéro de ligne>
<ordre RETURN> ::= 'RETURN'
<ordre STØP> ::= "STØP"
<ordre END> ::= 'END'
<FØR> ::= 'FØR'
<TØ> ::= 'TØ'
<STEP> ::= 'STEP'
<FØR partie gauche 3> ::= <FØR> <partie gauche 3>
<FØR partie gauche 3 expression> ::= <FØR partie gauche 3>
                                     <expression>
<TØ expression> ::= <TØ> <expression>
<STEP expression> ::= <STEP> <expression>
<ordre FØR> ::= <FØR partie gauche 3 expression> <TØ expression> |
        <FØR partie gauche 3 expression> <TØ expression>
        <STEP expression>
<NEXT> ::= 'NEXT'
<ordre NEXT> ::= <NEXT> <variable simple>
<DATA> ::= 'DATA'
<ordre DATA> ::= <DATA> <constante> | <ordre DATA> <virgule>
                                      <constante>
<READ> ::= 'READ'
<ordre READ> ::= <READ> <liste de variables>
<INPUT> ::= 'INPUT'
<ordre INPUT> ::= <INPUT> <liste de variables>
```

```
<PRINT>  ::= 'PRINT'
<point-virgule> ::= ';'
<print 2>  ::=<impression><point-virgule>|<impression><virgule>
<impression> ::= <PRINT><expression>| <print 2><expression>|
                 <PRINT><chaîne impression>| <print 2>
                                          <chaîne impression>
<print 1> ::= <PRINT>|<print 2>
<chaîne impression> ::= <"chaîne de caractères">
<RESTØRE> ::=  'RESTØRE'
<ordre RESTØRE> ::= <RESTØRE>|<RESTØRE><numéro de ligne>
<DEF> ::= 'DEF'
<ordre DEF> ::= <DEF><fonction définie>
<REM> ::= 'REM'
<ordre REM> ::= <REM><chaîne de caractères>
<DIM> ::= 'DIM'
<ordre DIM> ::= <DIM><spécifications dimensions>|
                <ordre DIM><virgule><spécifications dimensions>
<MATREAD> ::= 'MATREAD'
<ordre MATREAD> ::= <MATREAD><tableau>|
                    <ordre MATREAD><virgule><tableau>
<MATINPUT> ::= 'MATINPUT'
<ordre MATINPUT> ::= <MATINPUT><tableau>|
                     <ordre MATINPUT><virgule><tableau>
<MATPRINT> ::= 'MATPRINT'
<matprint 1> ::=<MATPRINT><identificateur de tableau>|
               <matprint 2><identificateur de tableau>
<matprint 2> ::=<matprint 1><virgule>|<matprint 1><point virgule>
<ordre MATPRINT> ::= <matprint 1>|<matprint 2>
<MAT> ::= 'MAT'
<MAT tableau> ::= <MAT><tableau>
<MAT affectation> ::= <MAT tableau><signe d'affectation matricielle>
<ordre initialisation MAT> ::= <MAT affectation><fonction initiali-
                                                satrice>|
                               <MAT affectation><initialisation>
<ordre PRINT> ::= <print 1>|<print 2>
```

<ordre assignation MAT> ::= <MAT affectation><terme matriciel>|
<MAT affectation><identificateur de tableau>

<ordre MAT> ::= <ordre assignation MAT>|<ordre initialisation MAT>|
<ordre MATPRINT>|<ordre MATINPUT>|<ordre MATREAD>

<ordre BASIC> ::= <ordre MAT>|<ordre DIM>|<ordre REM>|<ordre DEF>|
<ordre RESTØRE>|<ordre PRINT>|<ordre INPUT>|
<ordre READ>|
<ordre DATA>|<ordre NEXT>|<ordre FØR>|<ordre END>|
<ordre STØP>|<ordre RETURN>|<ordre GØSUB>|
<ordre GØTØ>|<ordre IF>|<ordre LET>

<INSTRUCTION> ::= 'd' <ordre BASIC> 'f'

V-1-2 - Remarques

On peut remarquer que généralement, les parties droites des productions sont limitées à deux termes, ceci pour faciliter la recherche dans la pile d'analyse, ainsi que nous le verrons bientôt, néanmoins, il existe des parties droites à 3 termes, c'est le cas pour les définitions récursives du genre :

<liste de variables> ::= <variable>|<liste de variables>
<virgule> <variable>

ou pour l'ordre FØR qui peut comporter 3 termes si le pas de bouclage est précisé.

Il apparaît un certain nombre de numéros cerclés à gauche de certaines productions, ces numéros sont en fait des indicateurs de contexte.

Nous savons qu'il peut y avoir certaines ambiguïtés sur la signification d'éléments terminaux tels que : '=', '*', etc...,

l'indicateur de contexte est un moyen pratique pour les lever. Nous avons introduit d'autres indicateurs de ce type pour satisfaire à la condition de "non-recul".

V-1-3 - Indicateurs de contexte

V-1-3-1 - Fonctionnement

①	Autorisé pour	<MAT>, <DIM>, <MATINPUT>, <MATPRINT>, <MATREAD>
②	" "	<REM>, <">
③	" "	<MAT>
④	" "	<DATA>
⑤	" "	<READ>, <INPUT>
⑥		
⑦	" "	<MAT>
⑧	" "	<LET>, <FØR>, <DEF>
⑨	" "	<THEN>, <GØTØ>, <GØSUB>, <RESTØRE>
⑩	" "	<DIM>
⑪	" "	<MAT>, <MATINPUT>, <MATREAD>
⑫	" "	<FØR>
⑬	" "	<IF>
⑭	" "	<DEF>

V-1-3-2 - Justifications

IND1 Un identificateur de tableau ne peut exister que dans le contexte des instructions matricielles, l'utilisation d'un tableau dans une expression arithmétique simple se faisant par le biais d'une variable indicée.

IND2 Dans un ordre REMark (Commentaire), on peut utiliser n'importe quel caractère, ainsi que dans une chaîne littérale qu'on veut imprimer.

IND3 Il faut distinguer la nature du signe + (ou -) qui peut opérer soit sur des variables, soit sur des matrices.

IND4 Les constantes numériques ne peuvent apparaître que dans les blocs DATA. Partout ailleurs, on introduit des nombres.

IND5 Les listes de variables ne peuvent apparaître que dans les ordres de lecture multiple dans les blocs DATA, ou dans les ordres d'entrée multiple de nombres au clavier.

IND6 C'est un indicateur de contexte limité, dont nous verrons l'utilité plus tard.

IND7 Pour distinguer le signe d'affectation matricielle.

IND8 Pour distinguer le signe d'égalité habituel.

IND9 Permet de transformer un entier en un numéro de ligne.

IND10 Permet de transformer un entier en une quantité dans le cadre des dimensionnements de tableaux.

IND11 Permet de transformer une expression en dimension dans le cadre de certaines instructions matricielles.

IND12 Pour séparer < partie gauche 3 > et < partie gauche 2 >, il faut éviter l'affectation multiple dans un ordre FØR.

IND13 L'ordre IF est le seul qui tolère les caractères de comparaison.

IND14 Lors d'une définition de fonction interne, on doit avoir une variable simple en argument.

En fait, ces justifications apparaissent bien plus évidentes quand on transforme la représentation de la grammaire de façon à faire apparaître les réductions. Ces indicateurs de contexte ne sont pas liés à la grammaire qui est "context-free", mais à la méthode d'analyse choisie, pour rendre plus performant le processus d'analyse.

V-1-4 - Codage des éléments syntaxiques

<INSTRUCTION>	<1>	<ordre PRINT>	<26>
<ordre BASIC>	<2>	<chaîne impression>	<27>
<ordre MAT>	<3>	<print 1>	<28>
<ordre assignation MAT>	<4>	<impression>	<29>
<ordre initialisation MAT>	<5>	<print 2>	<30>
<MAT affectation>	<6>	<point-virgule>	<31>
<MAT tableau>	<7>	<PRINT>	<32>
<MAT>	<8>	<ordre INPUT>	<33>
<ordre MATPRINT>	<9>	<INPUT>	<34>
<matprint 2>	<10>	<ordre READ>	<35>
<matprint 1>	<11>	<READ>	<36>
<MATPRINT>	<12>	<ordre DATA>	<37>
<ordre MATINPUT>	<13>	<DATA>	<38>
<MATINPUT>	<14>	<ordre NEXT>	<39>
<ordre MATREAD>	<15>	<NEXT>	<40>
<MATREAD>	<16>	<ordre FØR>	<41>
<ordre DIM>	<17>	<STEP expression>	<42>
<DIM>	<18>	<TØ expression>	<43>
<ordre REM>	<19>	<FØR partie gauche 3 expression>	<44>
<REM>	<20>	<FØR partie gauche 3>	<45>
<ordre DEF>	<21>	<STEP>	<46>
<relation>	<22>	<TØ>	<47>
<DEF>	<23>	<FØR>	<48>
<ordre RESTØRE>	<24>	<ordre END>	<49>
<RESTØRE>	<25>	<ordre STØP>	<50>

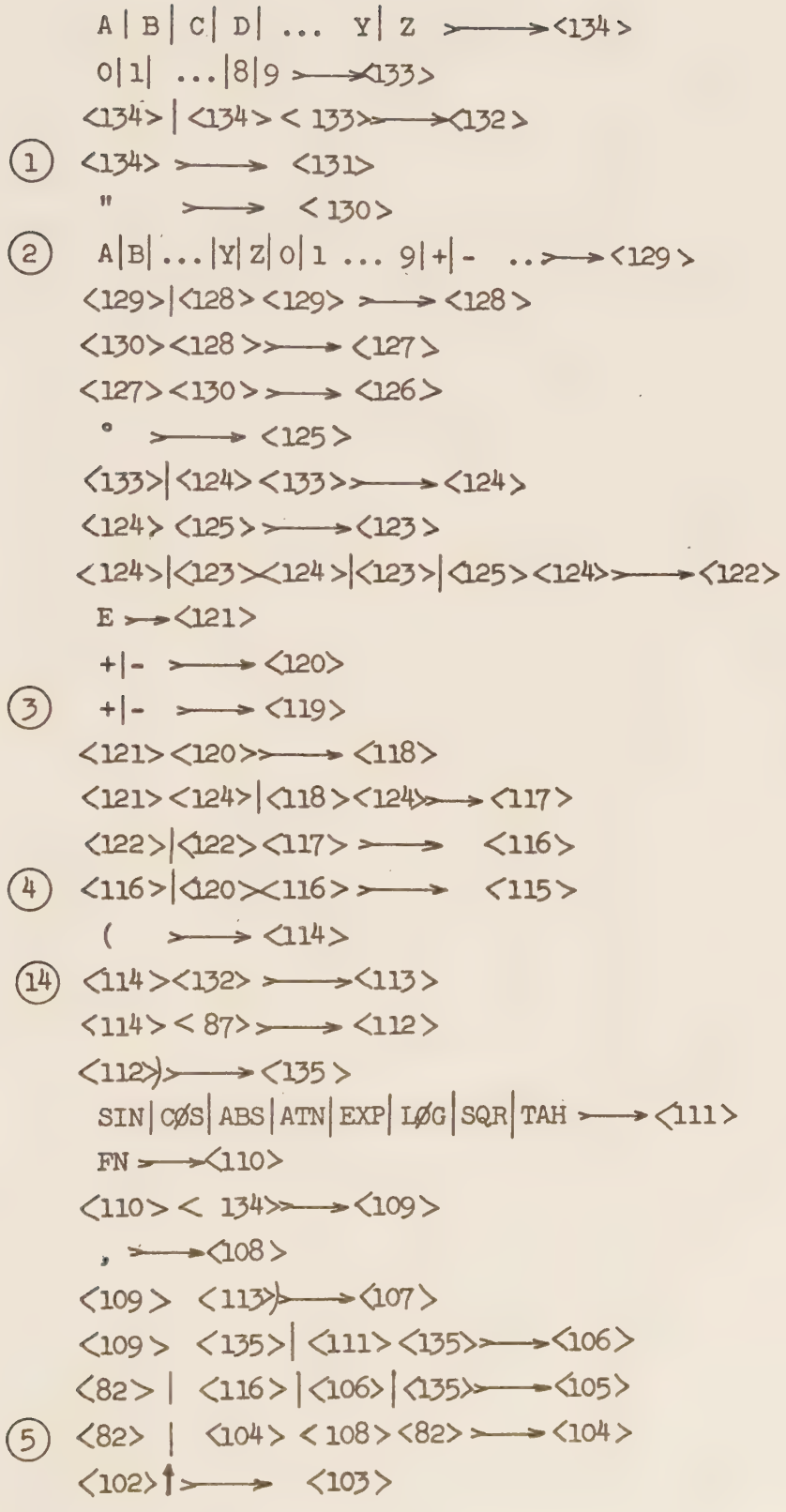
<ordre RETURN> <51>
<ordre GØSUB> <52>
<GØSUB> <53>
<ordre GØTØ> <54>
<GØTØ> <55>
<ordre IF> <56>
<THEN numéro de ligne> <57>
<expression IF> <58>
<THEN> <59>
<IF> <60>
<ordre LET> <61>
<LET> <62>
<terme matriciel> <63>
<opérateur matriciel> <64>
<initialisation> <65>
<fonction initialisatrice (> <66>
<fonction matricielle> <67>
<fonction initialisatrice> <68>
<tableau> <69>
<identificateur de tableau (> <70>
<dimensions> <71>
<dimension,> <72>
<spécifications dimensions> <73>
<quantité 1> <74>
<quantité,> <75>
<quantité> <76>
<fonction définie> <77>
<fonction 1> <78>
<partie gauche 3> <79>
<numéro de ligne> <80>
<partie gauche 2> <81>
<variable> <82>
<variable indicée> <83>
<liste d'indices> <84>
<liste d'indices,> <85>
<lettre (> <86>
<expression> <87>
<partie gauche 1> <88>
<signe égal> <89>
<signe d'affectation matricielle> <90>
<signe de relation> <91>
<minmax> <92>
<MinMax> <93>
<mineur majeur> <94>
<somme> <95>
<somme S> <96>
<terme> <97>
<opérateur de multiplication> <98>
<opérateur d'addition> <99>
<multiplication matricielle> <100>
<signe de multiplication> <101>
<facteur> <102>

<opérateur d'exponentiation>	<103>	<signe addition>	<120>
<liste de variables>	<104>	<exposant>	<121>
<primaire>	<105>	<nombre décimal>	<122>
<fonction 2>	<106>	<entier pointé>	<123>
<fonction 1>	<107>	<entier>	<124>
<virgule>	<108>	<point>	<125>
<identificateur de fonction>	<109>	<"chaîne de caractères">	<126>
<définition>	<110>	<"chaîne de caractères>	<127>
<fonction prédéfinie>	<111>	<chaîne de caractères>	<128>
<(expression>	<112>	<caractère>	<129>
<(variable simple>	<113>	<guillemet>	<130>
<ouverture de parenthèse>	<114>	<identificateur de tab.>	<131>
<constante>	<115>	<variable simple>	<132>
<nombre>	<116>	<chiffre>	<133>
<partie exposant>	<117>	<lettre>	<134>
<opérateur exposant>	<118>	<(expression)>	<135>
<addition matricielle>	<119>	<fonction matricielle (>	<136>

On s'aperçoit que cette grammaire a un vocabulaire non-terminal de 136 éléments. ($|V_N| = 136$). C'est un nombre de même ordre de grandeur que pour ALGØL ou FØRTRAN. On peut donc, de ce point de vue, assimiler le degré d'évolution de BASIC, avec celui des deux langages cités.

V-1-5 - Règles de réduction

On peut maintenant transformer la grammaire du § V-1-1 pour faire apparaître l'ensemble des règles de réduction. On utilise directement les codes décrits ci-dessus.


A | B | C | D | ... Y | Z → <134>
0 | 1 | ... | 8 | 9 → <133>
<134> | <134> < 133> → <132>
① <134> → <131>
" → < 130>
② A | B | ... | Y | Z | 0 | 1 ... 9 | + | - .. → <129>
<129> | <128> <129> → <128>
<130> <128> → <127>
<127> <130> → <126>
° → <125>
<133> | <124> <133> → <124>
<124> <125> → <123>
<124> | <123> <124> | <123> | <125> <124> → <122>
E → <121>
+ | - → <120>
③ + | - → <119>
<121> <120> → <118>
<121> <124> | <118> <124> → <117>
<122> | <122> <117> → <116>
④ <116> | <120> <116> → <115>
(→ <114>
⑭ <114> <132> → <113>
<114> < 87> → <112>
<112> → <135>
SIN | CØS | ABS | ATN | EXP | LØG | SQR | TAH → <111>
FN → <110>
<110> < 134> → <109>
, → <108>
<109> <113> → <107>
<109> <135> | <111> <135> → <106>
<82> | <116> | <106> | <135> → <105>
⑤ <82> | <104> < 108> <82> → <104>
<102> ↑ → <103>

<105> | <103> <105> ⟶ <102>

* | / ⟶ <101>

(3) * ⟶ <100>

<96> <120> ⟶ <99>

<97> <101> ⟶ <98>

<120> <102> | <98> <102> | <102> ⟶ <97>

<97> | <99> <97> ⟶ <96>

<96> ⟶ <95>

MIN/MAX ⟶ <94>

<92> <94> ⟶ <93>

<95> | <93> <95> ⟶ <92>

(13) < | <= | = | <> | >= | > ⟶ <91>

(7) = ⟶ <90>

(8) = ⟶ <89>

<92> <91> ⟶ <88>

<88> <92> ⟶ <22>

<92> ⟶ <87>

<134> <114> ⟶ <86>

<87> <108> ⟶ <85>

<85> <87> | <87> ⟶ <84>

<86> <84>) ⟶ <83>

<132> | <83> ⟶ <82>

<82> <89> | <81> <82> <89> ⟶ <81>

(9) <124> ⟶ <80>

(12) <132> <89> ⟶ <79>

<107> <89> ⟶ <78>

<78> <87> ⟶ <77>

(10) <124> ⟶ <76>

<76> <108> ⟶ <75>

<75> <76> ⟶ <74>

<131> <114> ⟶ <70>

<70> <74>) | <70> <76>) ⟶ <73>

(11) <87> <108> ⟶ <72>

(11) <72> <87> | <87> ⟶ <71>

<70><71>) | <131> → <69>

ZER | CØN | IDN → <68>

INV | TRN → <67>

<67> <114> → <136>

<68> <114> → <66>

<66> <71>) → <65>

<135><100> | <131> < 119> | <131> < 100> → <64>

<64> <131> | <136> < 131>) → <63>

LET → <62>

<62> <81> < 87> → <61>

IF → <60>

THEN → <59>

<60> < 22> → <58>

<59> <80> → <57>

<58> <57> → <56>

GØTØ → <55>

<55> < 80> → <54>

GØSUB → <53>

<53> < 80> → <52>

RETURN → <51>

STØP → <50>

END → <49>

FØR → <48>

TØ → <47>

STEP → <46>

<48> <79> → <45>

<45> <87> → <44>

<47> <87> → <43>

<46> <87> → <42>

<44> <43> | <44> <43> <42> → <41>

NEXT → <40>

<40> < 132> → <39>

DATA → <38>

<38> <115> | <37> < 108> < 115> → < 37>

READ → <36>

<36> <104>⟶<35>

INPUT ⟶ <34>

<34> <104>⟶<33>

PRINT ⟶ <32>

; ⟶ <31>

<29><31> | <29><108>⟶<30>

<32><87>|<30><87>|<32><27>|<30><27>⟶<29>

<32><29>⟶<28>

<126> ⟶ <27>

<28><30>⟶<26>

RESTØRE⟶<25>

<25>|<25><80>⟶<24>

DEF⟶<23>

<23><77>⟶<21>

REM⟶<20>

<20><128>⟶<19>

DIM⟶<18>

<18><73>|<17><108><73>⟶<17>

MATREAD⟶<16>

<16><69>|<15><108><69>⟶<15>

MATINPUT⟶<14>

<14><69>|<13><108><69>⟶<13>

MATPRINT⟶<12>

<12><131>|<10><131>⟶<11>

<11><108>|<11><31>⟶<10>

<11>|<10>⟶<9>

MAT⟶<8>

<8><131>⟶<7>

<7><90>⟶<6>

<6><68>|<6><65>⟶<5>

<6><63>|<6><131>⟶<4>

<4>|<5>|<9>|<13>|<15>⟶<3>

<3>|<17>|<19>|<21>|<24>|<26>|<33>|<35>|<37>

<39>|<41>|<49>|<50>|<51>|<52>|<54>|<56>|<61>⟶<2>

d <2> f⟶<1>

V-1-6 - Remarques

On constate que l'axiome est < INSTRUCTION > et non < programme >, comme on en a l'habitude. Ceci provient du fait que chaque ligne du programme BASIC, forme en soi une entité, indépendante des autres lignes (sauf pour quelques types d'instructions), chaque ligne peut donc être analysée isolément, et l'axiome est alors < instruction > .

V-2 - CONTRAINTES SUPPLEMENTAIRES DE BASIC

Pour un langage conversationnel du type de BASIC, on a l'habitude d'entrer un programme par l'intermédiaire d'un télétype, nous pouvons simuler le télétype en entrée à l'aide du clavier alpha-numérique dont nous disposons.

Chaque ligne doit être impérativement numérotée entre 1 et 9999.

Chaque ligne peut comporter 72 caractères au plus.

Les lignes du programme peuvent être entrées dans un ordre quelconque, le compilateur est équipé d'une séquence de tri des instructions qui rétablit le programme dans l'ordre de numérotation.

L'étiquette de numéro le plus élevé doit faire référence à une instruction END.

Les réponses du compilateur se font normalement par télé-type, nous les ferons passer par l'imprimante, en simulant du mieux possible le comportement d'un télétype.

V-3 - GRAPHE D'ORDRE PARTIEL

Une grammaire aussi développée ne permet pas qu'on trace son graphe, car il devient vite inextricable. De plus, nous avons vu au chapitre précédent que seul le graphe d'ordre partiel joue un rôle essentiel.

Ce graphe, figure V-1, appelle quelques explications, les indicateurs de contexte apparaissent dans les liaisons représentées (n) ~~xxxxxx~~, où n est le numéro de l'indicateur mis en jeu, les liaisons du type --- symbolisent le fait que l'élément du bas est obtenu de l'élément du haut avec une séparation due à un caractère délimiteur (essentiellement : '↑', ')' et 'f').

Un élément du type (<N>) symbolise un des ordres BASIC, et (<N>) symbolise une des formes des ordres matriciels regroupés en (<3>).

Tous les éléments (<N>) se regroupent dans ( ).

Les structures globalement récursives, comme les expressions entre parenthèses, ou les variables indicées, sont repérées à leur obtention par [<N>], et à leur utilisation par [<N>].

L'axiome, quant à lui, apparaît comme <1>.

Ce graphe montre qu'il n'y a plus une entrée privilégiée comme dans l'exemple que nous avions choisi, au chapitre précédent pour illustrer la méthode.

Il faut considérer les véritables entrées du graphe, comme étant <identificateur de tableau> ^{131}, <variable simple> ^{132}, <entier> ^{124}, <fonction 2> ^{106}. Ces entrées représentent en fait

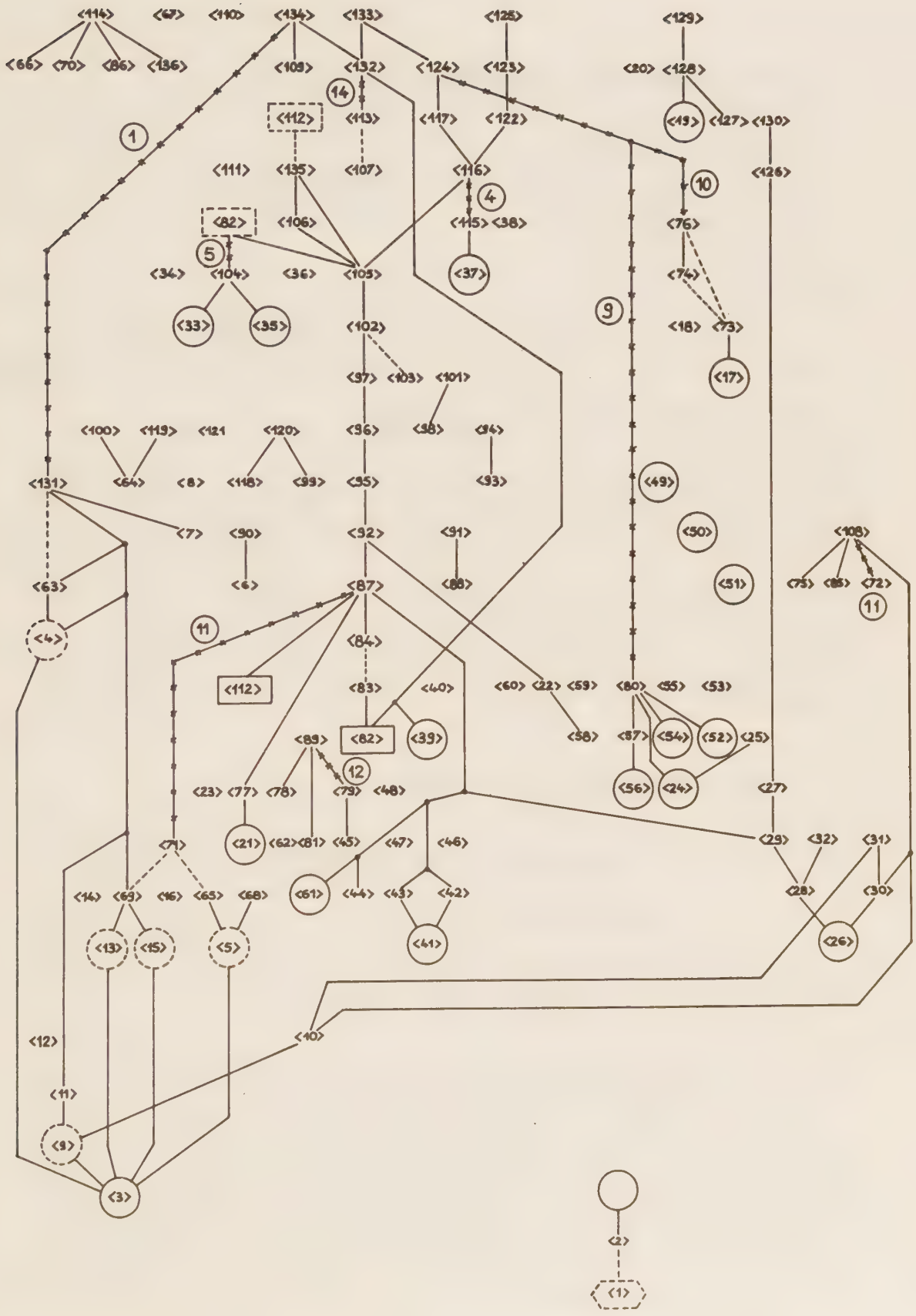

<114> <67> <110> <134> <133> <125> <129>
<66> <70> <86> <136> <109> <132> <124> <123> <20> <128>
14
<112> <113> <117> <122> <19> <127> <130>
1
<111> <135> <107> <116> <126>
10
4
<82> <106> <115> <38> <76>
5
<34> <104> <36> <105> <37> <74>
<33> <35> 9 <18> <73>
<102>
<17>
<97> <103> <101>
<100> <119> <121 <120> <96> <98> <94>
<131> <64> <8> <118> <99> <95> <93> <49>
<7> <90> <92> <91> <50> <108>
<63> <6> <87> <88> <51> <75> <85> <72>
11
<4>
11
<84>
<112> <83> <40> <60> <22> <59> <80> <55> <53>
<89> <82> <39> <58> <87> <54> <52> <25>
12
<23> <77> <78> <79> <48> <56> <24> <27>
<21> <62> <81> <45> <47> <46> <29> <32> <31>
<71>
<14> <69> <16> <65> <68> <61> <44> <43> <42> <28> <30>
<13> <15> <5> <41> <26>
<12> <10>
<11>
<9>
<3>
<2>
<1>

Figure V-1

les types d'opérandes sur lesquels on travaille :

- tableaux ou matrices
- variables simples (ou élément de matrice = variable indicée)
- quantités numériques
- fonctions

Nous classons de moindre importance ici, l'entrée $<$caractère$>^{128}$, car il n'y a pas à proprement parler de $<$ variable chaîne de caractères $>$ dans ce langage.

Remarquons que le graphe figure IV-12 est inclus dans celui-ci au niveau de la structure graphique comprise entre les noeuds $<105>$ et $<87>$, sauf qu'il y a, ici, des éléments supplémentaires provenant de l'introduction des symboles MIN et MAX. Ce fait que nous soulignons signifie que le langage d'expressions arithmétiques est un sous-langage du BASIC (en fait de tout langage de programmation scientifique).

V-4 - CARACTERISTIQUES DE L'ANALYSEUR

V-4-1 - Numéro de ligne - Classement des instructions

La vérification de la présence et de la validité du numéro de ligne est confiée au module de l'analyseur qui permet la frappe au clavier alphanumérique des instructions.

Le passage en analyse de l'instruction proposée fournit l'information dont on a besoin, ou bien l'instruction est syntaxiquement correcte et on l'intègre à l'ensemble des instructions déjà correctes, en procédant au classement, ou bien l'instruction présente une erreur de syntaxe et on la rejette purement et simplement, en imprimant le numéro de l'erreur détectée, il n'y a pas alors besoin de refaire le classement.

Les organigrammes des séquences de prise en charge des instructions et de classement figurent en annexe B.

V-4-2 - Mots-clés

On a vu que le positionnement des indicateurs de contexte dépend du type d'instruction rencontrée, c'est-à-dire du mot-clé qui se trouve immédiatement après le numéro de ligne.

En fait, les mots-clés forment un ensemble assez vaste qui comprend : les noms d'instructions, les noms de fonctions prédéfinies, les symboles MIN et MAX, et quelques autres mots réservés aux fonctions matricielles.

Les mots-clés se composent de suites de lettres plus ou moins longues (2 à 8 lettres). C'est donc au niveau de la détection de l'élément terminal <lettre> qu'il importe de reconnaître les mots-clés.

Sachant qu'en BASIC, toute suite d'au moins 2 lettres différente de FN, doit être un mot-clé, il suffit de tester si l'élément terminal suivant est une lettre, étant donné qu'on a déjà reconnu l'élément terminal courant (c'est une lettre).

Il faut maintenant reconnaître si la suite des lettres qui se présentent est un mot-clé autorisé, on consulte pour cela la liste enchaînée des mots-clés du langage créée par programme (voir annexe A), caractère par caractère. La consultation négative se termine par un code erreur, la consultation positive se termine par l'obtention du numéro attribué au mot-clé. Ce numéro permet par consultation d'une table de branchement, d'aller exécuter la séquence de traitement du mot-clé, où, en particulier, on positionne les indicateurs de contexte.

V-4-3 - Position d'analyse - Pile d'analyse

La figure V-2 fait apparaître pour première instruction, le chargement d'une position dénommée AN, à partir de la position de la chaîne d'entrée repérée par le pointeur PTR, en d'autres termes, on charge le caractère courant ((PTR)), dans la position d'analyse AN, on fait évoluer le pointeur, qui contiendra alors l'adresse du caractère suivant.

La position d'analyse joue bien sûr un rôle très important, celui que lui confère sa représentation du caractère terminal courant. La consultation de AN, figure V-2, permet le branchement correct aux différentes routines de traitement des éléments terminaux. Elle permet aussi en cours d'analyse, de comparer par endroits le caractère courant à un caractère déterminé, de façon à faire un traitement plutôt qu'un autre, selon le résultat de la comparaison.

On sait qu'une réduction met en jeu deux éléments syntaxiques, plus rarement trois, il est donc important de pouvoir conserver les numéros de ces 2 ou 3 éléments syntaxiques, et ceci de façon évolutive, puisqu'on élimine les 2 ou 3 numéros pour les remplacer par le numéro de l'élément syntaxique obtenu par réduction. L'emploi d'une pile est un moyen très bien adapté à ce travail.

On place les numéros d'éléments syntaxiques en sommet de pile avec montée de pile correspondante, tant qu'il n'y a pas réduction ; on descend la pile et on y remplace les éléments réduits quand il y a réduction.

La pile d'analyse syntaxique, appelée P, est utilisée dans les organigrammes, sous la forme de l'élément de sommet ((P)), et des 3 éléments précédents ((P)-1), ((P)-2) et ((P)-3).

V-4-4 - Modules d'analyse syntaxique

On appelle module d'analyse syntaxique, toute séquence de traitement, qui, pour un élément syntaxique donné, explore toutes les règles de réduction où il entre.

On appelle module terminal, la séquence de traitement qui permet la prise en charge d'un élément terminal.

Il y a autant de modules terminaux que d'éléments terminaux, et autant de modules d'analyse syntaxique qu'il y a d'éléments syntaxiques susceptibles d'avoir un descendant dans le graphe d'ordre partiel.

V-4-5 - Aiguillage

Dans le cas d'une analyse sans indicateurs de contexte et avec un graphe d'ordre partiel à une seule entrée, tout élément terminal délimiteur (pratiquement, différent des lettres), n'entrant pas directement en jeu dans une réduction, doit permettre les réductions successives de l'élément syntaxique qui le précède, et dont l'état initial est <variable> (si on se limite à un langage où ne figurent que des variables). La méthode consiste alors, après mise en forme du caractère délimiteur dans le module terminal correspondant, à se brancher au module d'analyse syntaxique dont le nom est (variable.)

Dans notre cas, ce n'est pas aussi simple, car il y a plusieurs entrées dans le graphe d'ordre partiel, il y a des indicateurs de contexte, qui nous permettent d'obtenir des informations nécessaires à des réductions conditionnelles, dont il ne faut pas perdre le bénéfice, ce qui fait que le module où il faut reprendre après détection d'un caractère délimiteur, n'est pas toujours le même.

La solution choisie consiste en l'emploi d'une position d'aiguillage. Dans les modules d'analyse syntaxique, on positionne l'aiguillage dans un état tel qu'après le passage d'un caractère délimiteur, on se branche au module de reprise voulu.

V-4-6 - Ambiguïté de certains éléments terminaux

On sait qu'il existe certains éléments terminaux qui peuvent avoir, a priori, des significations différentes, par exemple le caractère * peut avoir 2 significations. Ceci n'est pas grave, puisque nous savons qu'on lève l'ambiguïté facilement à l'aide d'un indicateur de contexte. Si l'on attribue maintenant un code spécial, n'appartenant pas au jeu de code normal, au caractère qu'on veut spécialiser, il n'y a plus de problèmes dans la suite de l'analyse.

Les caractères qu'on est ainsi amené à coder spécialement sont :

* matriciel, + | - matriciel, signe de relation, affectation matricielle.

Il faut également coder des caractères terminaux n'appartenant pas au jeu de caractères, tels les délimiteurs de début d et de fin f, ainsi que les symboles critères MIN | MAX.

V-4-7 - Arbre d'analyse syntaxique

Même si en exploitation, nous n'avons pas besoin de l'arbre d'analyse syntaxique, il nous a semblé intéressant, dans le cadre de cette étude, de montrer comment on pouvait obtenir cet arbre.

A chaque endroit où il y a une réduction, nous savons parfaitement quels sont les éléments qui sont réduits, donc leur nombre, et l'élément obtenu par réduction, ce qui se traduit par une

position de graphe, où l'élément de réduction est un noeud qui a n branches aboutissant aux n éléments réduits.

Un élément terminal étant une feuille, le nombre de branches pendantes est nul, par contre, pour chaque élément non-terminal, il y a au moins une branche pendante, plus généralement n ($n \leq 3$).

Nous renvoyons à l'annexe D pour les résultats pratiques obtenus après analyse de certaines instructions.

L'arbre d'analyse est stocké élément par élément dans une pile ascendante, dénommée ARB.

V-4-8 - Erreurs de syntaxe

Nous savons qu'un analyseur doit permettre des sorties erreur suffisamment significatives pour déterminer à quel endroit de l'instruction analysée il y a une erreur de syntaxe.

Les erreurs commises sont généralement des 3 catégories suivantes :

- deux délimiteurs se suivent.

- il n'y a pas de délimiteur à l'endroit où il devrait y en avoir un.

- emploi d'un délimiteur à la place d'un autre.

Si dans un module, on est amené à vérifier la présence dans la position d'analyse, d'un caractère qui ne s'y trouve pas, on peut être sûr qu'il y a erreur, et prévoir la sortie erreur correspondante.

Prenons un exemple :

Dans la définition d'une fonction interne, on doit avoir une variable simple comme argument : DEF FNA (X)

La variable simple doit être entre parenthèses, ce qui se traduit en partie par la règle de réduction suivante :

(14) <ouverture de parenthèse><variable simple>⟶<(variable simple>

l'indicateur IND14 étant positionné à 1 par le mot-clé DEF.

Dans le module d'analyse (variable simple) , la détection de IND14 à 1 nous amène à tester la présence de l'élément syntaxique <114> (parenthèse d'ouverture) dans la pile P, pour faire la réduction ci-dessus. Si nous ne trouvons pas <114> dans ((P)-2), nous pouvons sortir en erreur, puisque nous sommes d'ores et déjà sûrs que la structure est incorrecte.

Remarquons que dans le cas de structures très particulières, nous positionnons à 1 les indicateurs de contexte voulus à l'aide des mots-clés, nous faisons revenir ces indicateurs à 0 (généralement) à la détection de la fin de structure particulière, ceci pour revenir au cas normal. C'est particulièrement le cas de IND6 pour les structures de variables indicées.

V-4-9 - Mots-clés délimiteurs

Dans certains cas, certains mots-clés figurant dans le corps d'une instruction, doivent faire office de délimiteurs (en contexte postérieur) pour permettre l'achèvement de la vérification de la structure précédente.

Ces mots-clés particuliers sont :

THEN dans l'instruction IF xxxx THEN xxx

FØR et STEP dans l'instruction FØR xxxx TØ xxxx STEP xxxx

La méthode employée consiste à se débrancher en AIGUILLAGE après la reconnaissance d'un de ces mots-clés, en positionnant un indicateur spécialisé pour chacun d'eux, on descend dans le graphe d'ordre partiel et l'on passe alors par un test de présence du délimiteur mot-clé, (en fait, on teste la valeur de l'indicateur correspondant), en cas de réponse positive, on reprend le traitement normal du mot-clé, sinon c'est une sortie erreur.

V-5 - ORGANIGRAMMES DE L'ANALYSEUR

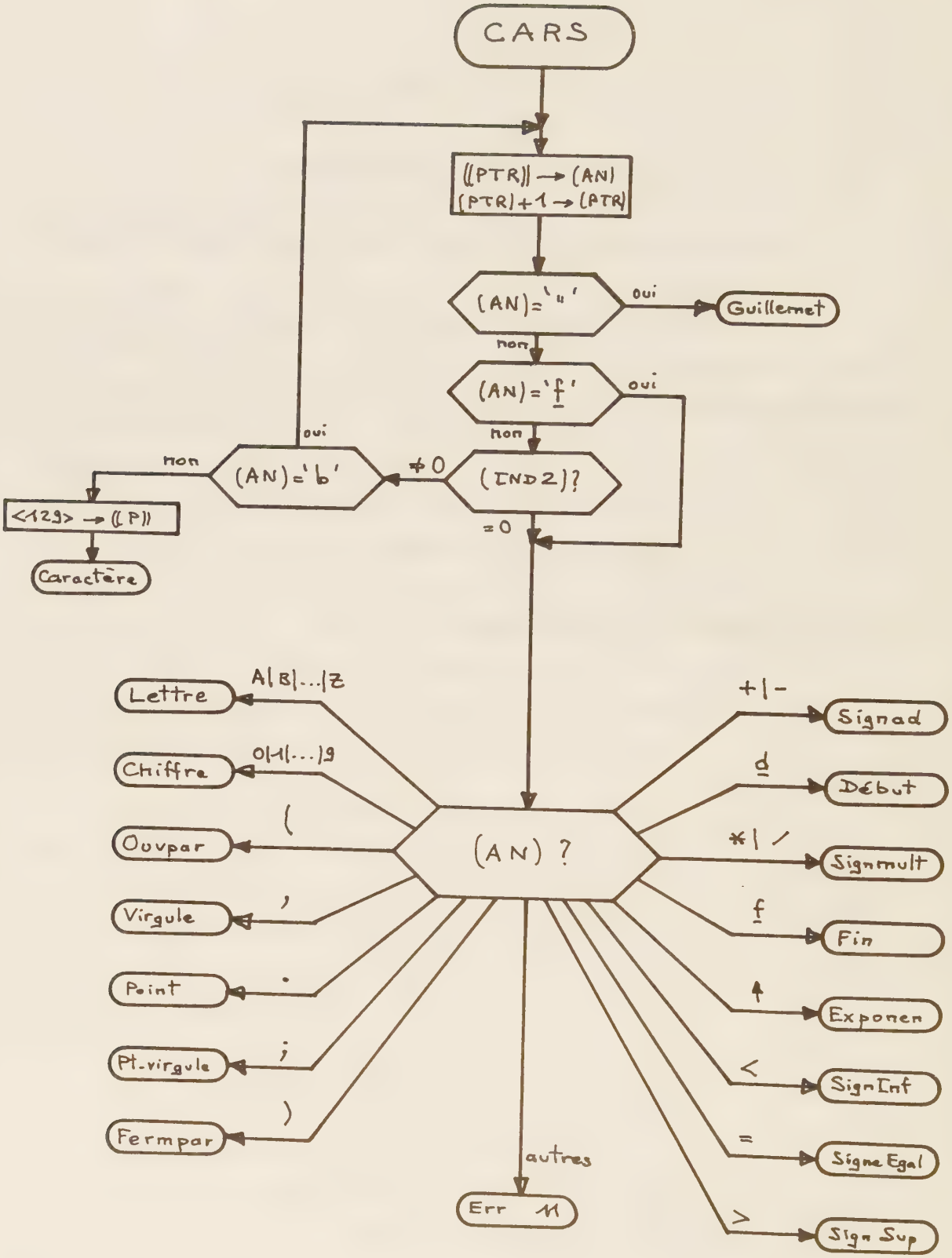

CARS
((PTR)) → (AN)
(PTR)+1 → (PTR)
(AN)='"'
oui
Guillemet
non
(AN)='f'
oui
non
(AN)='b'
oui
non
≠0
(IND2)?
=0
<129> → ((P))
Caractère
Lettre
A|B|...|Z
Chiffre
0|1|...|9
Ouvpar
(
Virgule
,
Point
.
Pt-virgule
;
Fermpar
)
(AN) ?
autres
Err 11
+|-
Signad
d
Début
*|/
Signmult
f
Fin
↑
Exponen
<
SignInf
=
Signe Egal
>
Sign Sup

Figure V-2

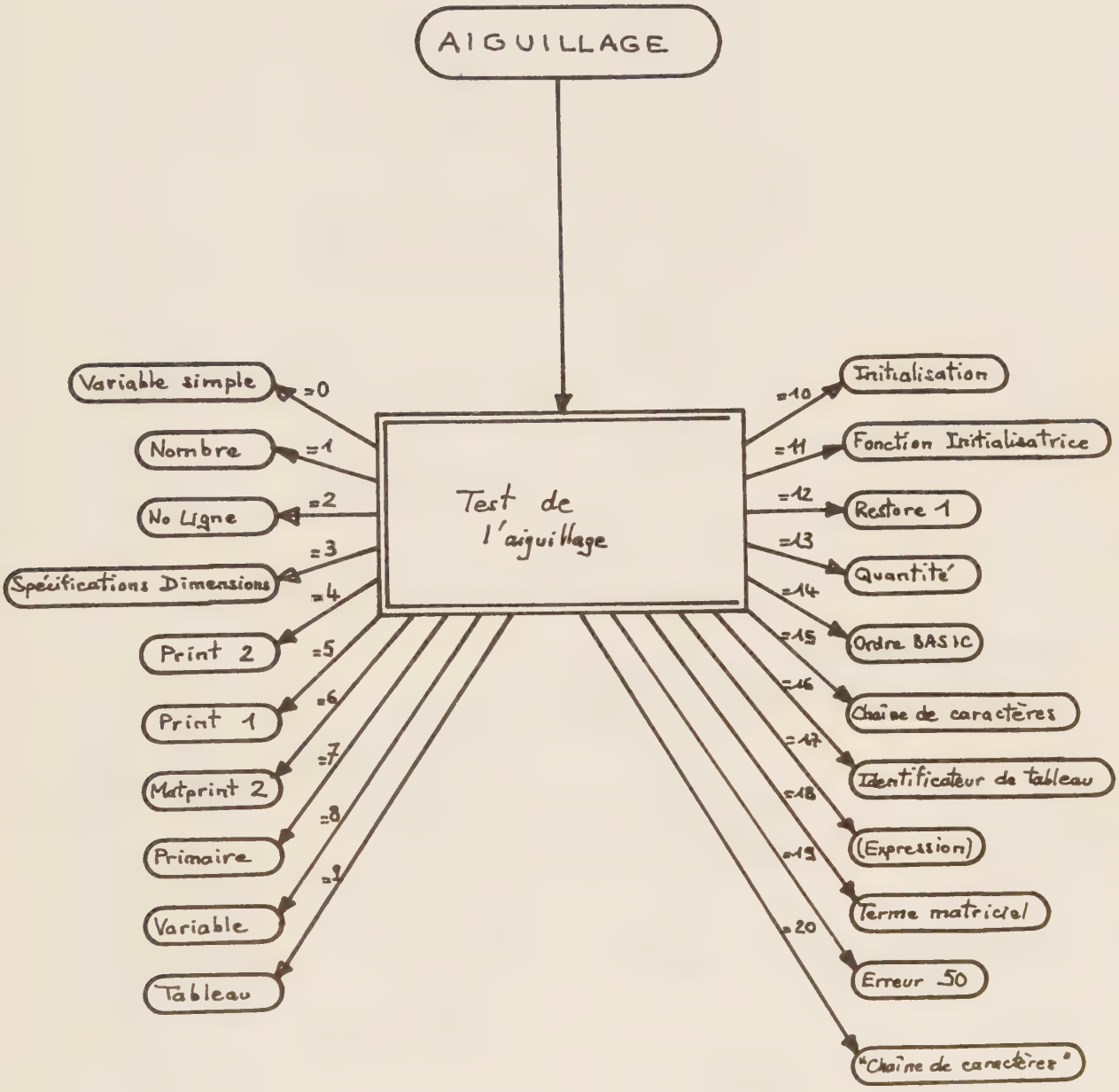

AIGUILLAGE
Test de l'aiguillage
Variable simple
=0
Nombre
=1
No Ligne
=2
Spécifications Dimensions
=3
Print 2
=4
Print 1
=5
Matprint 2
=6
Primaire
=7
Variable
=8
Tableau
=9
Initialisation
=10
Fonction Initialisatrice
=11
Restore 1
=12
Quantité
=13
Ordre BASIC
=14
Chaîne de caractères
=15
Identificateur de tableau
=16
(Expression)
=17
Terme matriciel
=18
Erreur 50
=19
"Chaîne de caractères"
=20

Figure V-3

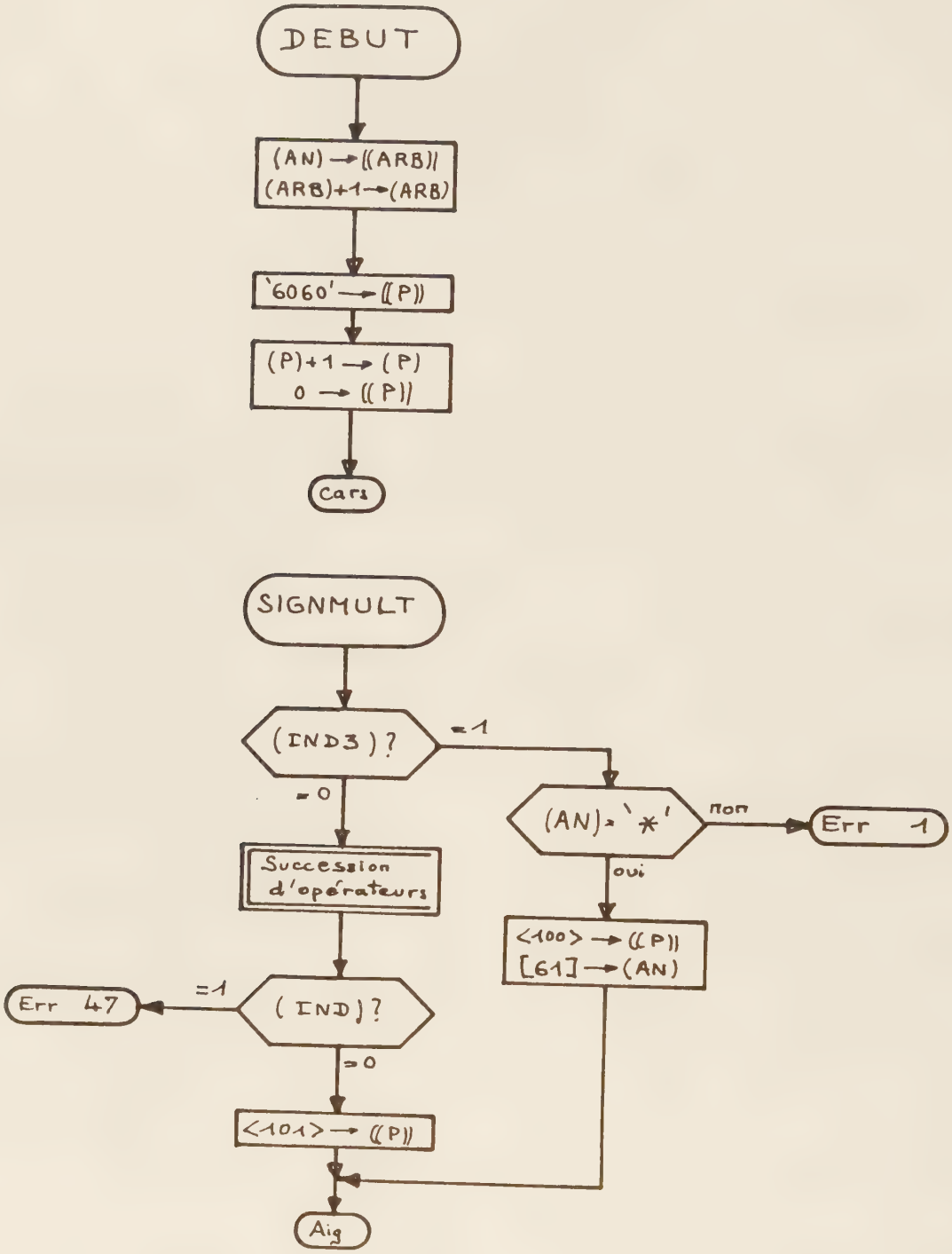

DEBUT
(AN) → ((ARB))
(ARB)+1 → (ARB)
'6060' → ((P))
(P)+1 → (P)
0 → ((P))
Cars
SIGNMULT
(IND3) ?
=1
=0
non
(AN) = '*'
Err 1
oui
Succession d'opérateurs
<100> → ((P))
[61] → (AN)
=1
Err 47
(IND) ?
=0
<101> → ((P))
Aig

Figure V-4

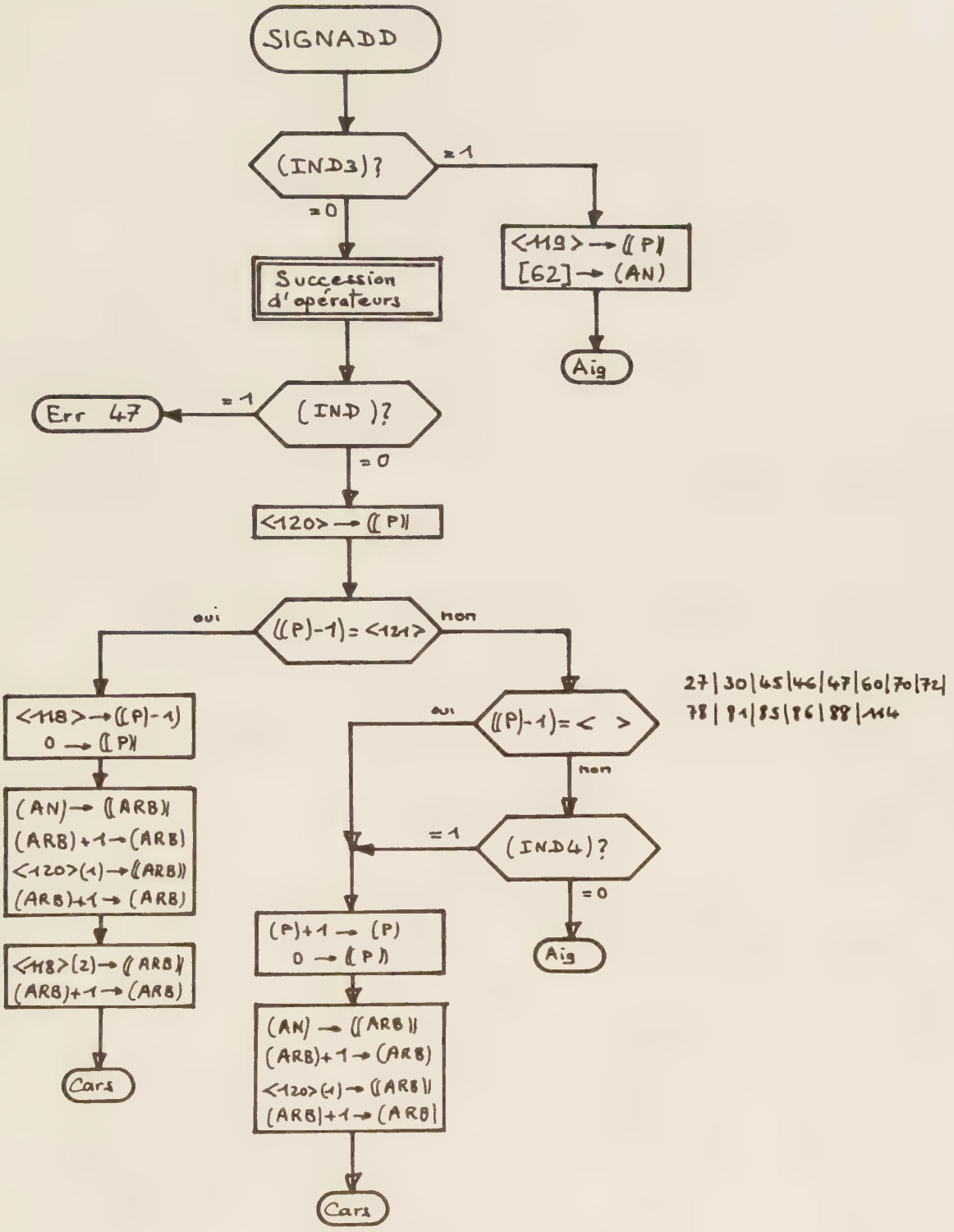

SIGNADD
(IND3)?
=1
=0
<119> → ((P))
[62] → (AN)
Aig
Succession d'opérateurs
(IND)?
=1
Err 47
=0
<120> → ((P))
((P)-1) = <121>
oui
non
<118> → ((P)-1)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<120>(1) → ((ARB))
(ARB)+1 → (ARB)
<118>(2) → ((ARB))
(ARB)+1 → (ARB)
Cars
((P)-1) = < >
oui
non
=1
(IND4)?
=0
Aig
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<120>(1) → ((ARB))
(ARB)+1 → (ARB)
Cars
27 | 30 | 45 | 46 | 47 | 60 | 70 | 72 |
78 | 81 | 85 | 86 | 88 | 114

Figure V-5

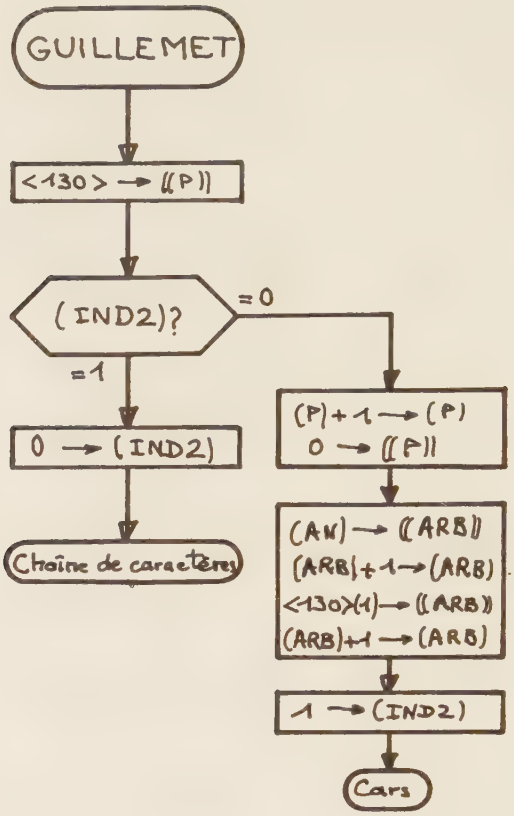


Figure V-6

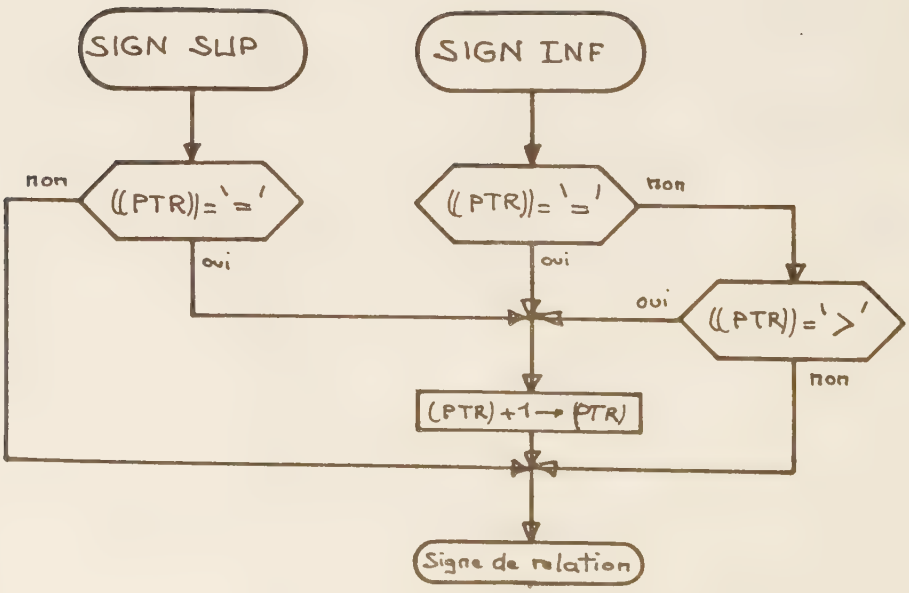


Figure V-7

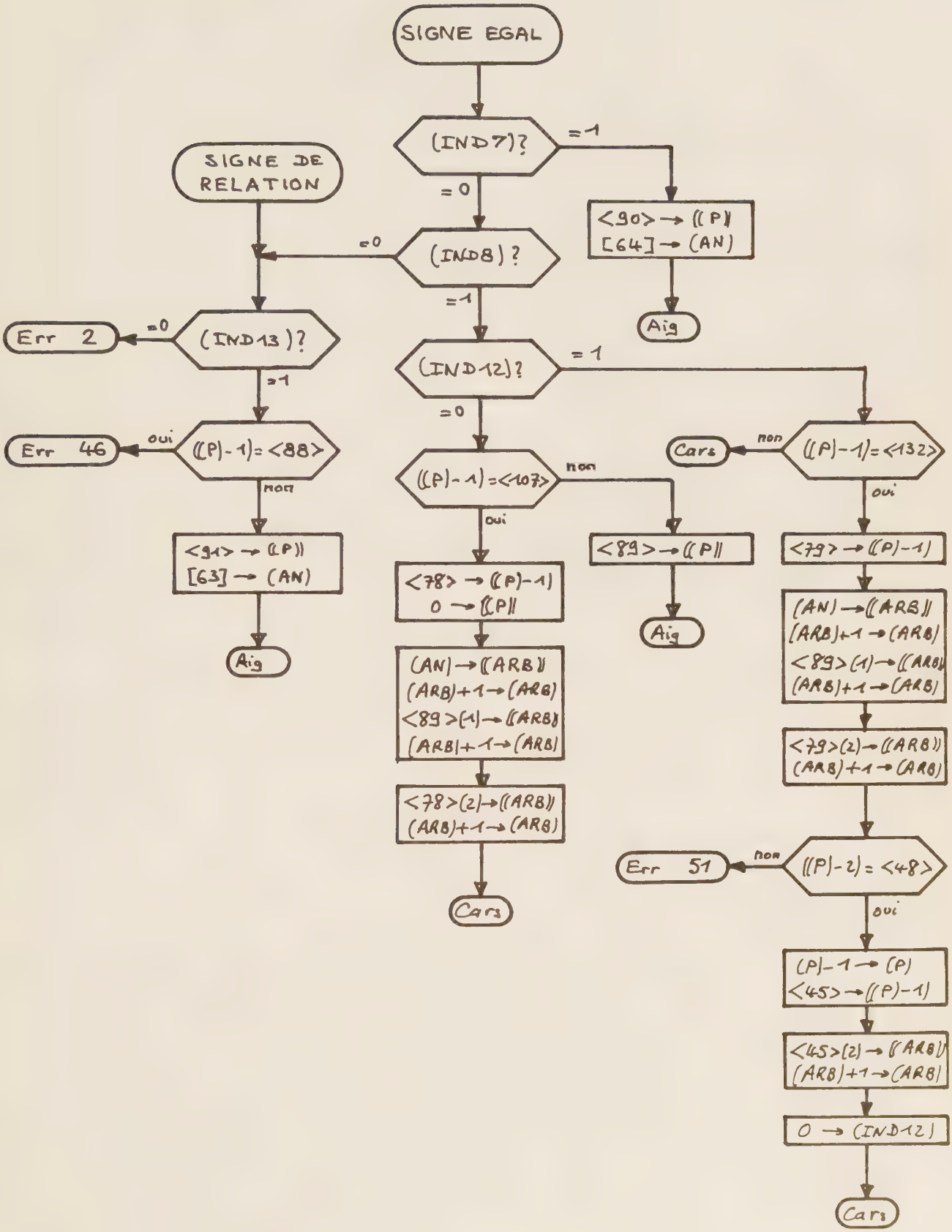
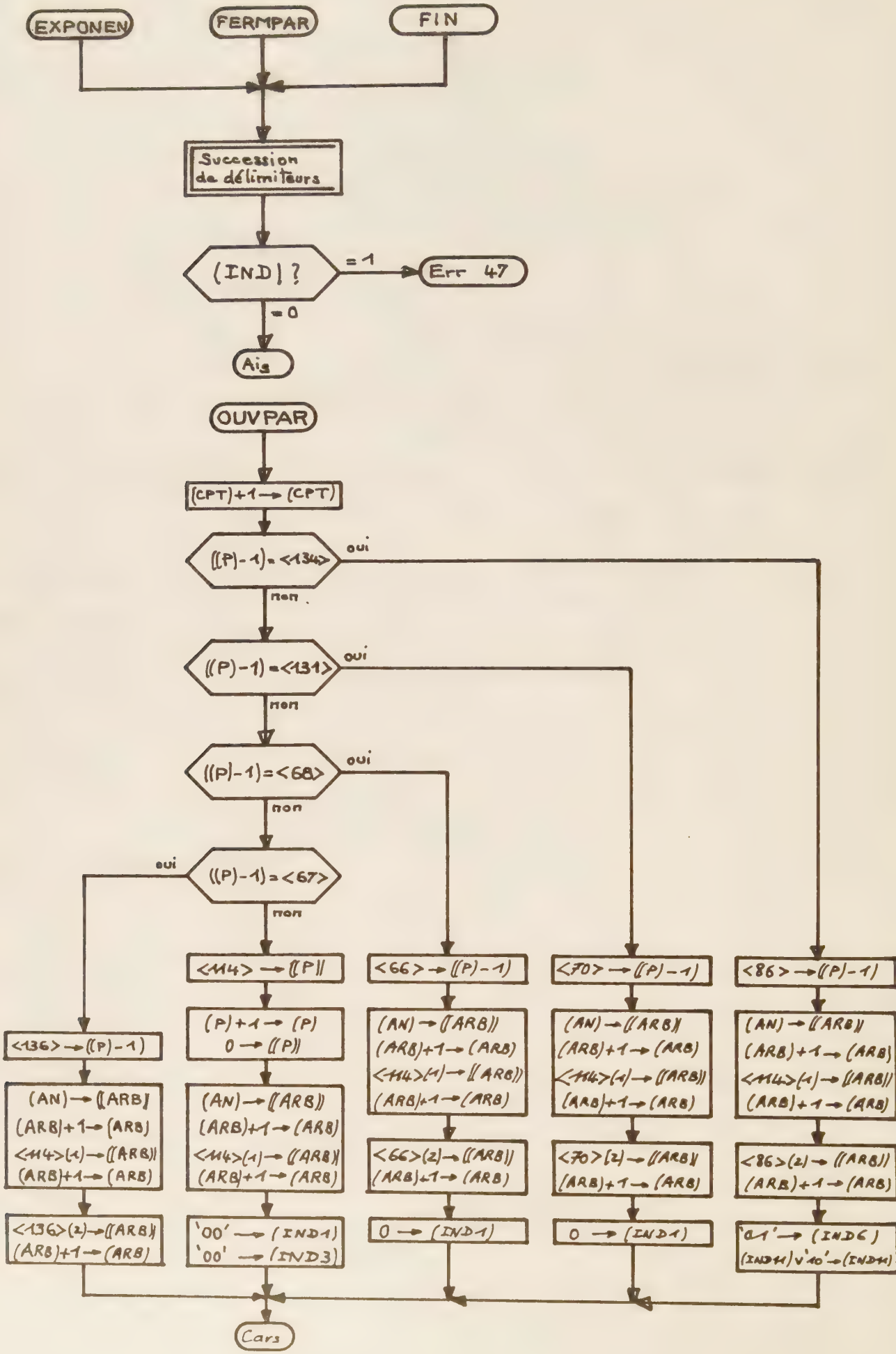

SIGNE EGAL
(IND7)?
=1
=0
SIGNE DE RELATION
<90> → ((P))
[64] → (AN)
Aig
=0
(IND8)?
=1
Err 2
=0
(IND13)?
>1
(IND12)?
= 1
=0
Err 46
oui
((P)-1)=<88>
non
Cars
non
((P)-1)=<132>
((P)-1)=<107>
non
oui
oui
<91> → ((P))
[63] → (AN)
<89> → ((P))
<79> → ((P)-1)
<78> → ((P)-1)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<89>(1) → ((ARB))
(ARB)+1 → (ARB)
Aig
Aig
(AN) → ((ARB))
(ARB)+1 → (ARB)
<89>(1) → ((ARB))
(ARB)+1 → (ARB)
<79>(2) → ((ARB))
(ARB)+1 → (ARB)
<78>(2) → ((ARB))
(ARB)+1 → (ARB)
Err 51
non
((P)-2)=<48>
oui
(P)-1 → (P)
<45> → ((P)-1)
<45>(2) → ((ARB))
(ARB)+1 → (ARB)
0 → (IND12)
Cars
Cars

Figure V-8


EXPONEN
FERMPAR
FIN
Succession de délimiteurs
(IND) ?
= 1
Err 47
= 0
Aig
OUVPAR
(CPT)+1 → (CPT)
((P)-1) = <134>
oui
non
((P)-1) = <131>
oui
non
((P)-1) = <68>
oui
non
((P)-1) = <67>
oui
non
<136> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<114>(1) → ((ARB))
(ARB)+1 → (ARB)
<136>(2) → ((ARB))
(ARB)+1 → (ARB)
<114> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<114>(1) → ((ARB))
(ARB)+1 → (ARB)
'00' → (IND1)
'00' → (IND3)
<66> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<114>(1) → ((ARB))
(ARB)+1 → (ARB)
<66>(2) → ((ARB))
(ARB)+1 → (ARB)
0 → (IND1)
<70> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<114>(1) → ((ARB))
(ARB)+1 → (ARB)
<70>(2) → ((ARB))
(ARB)+1 → (ARB)
0 → (IND1)
<86> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<114>(1) → ((ARB))
(ARB)+1 → (ARB)
<86>(2) → ((ARB))
(ARB)+1 → (ARB)
'01' → (IND6)
(IND11) v '10' → (IND11)
Cars

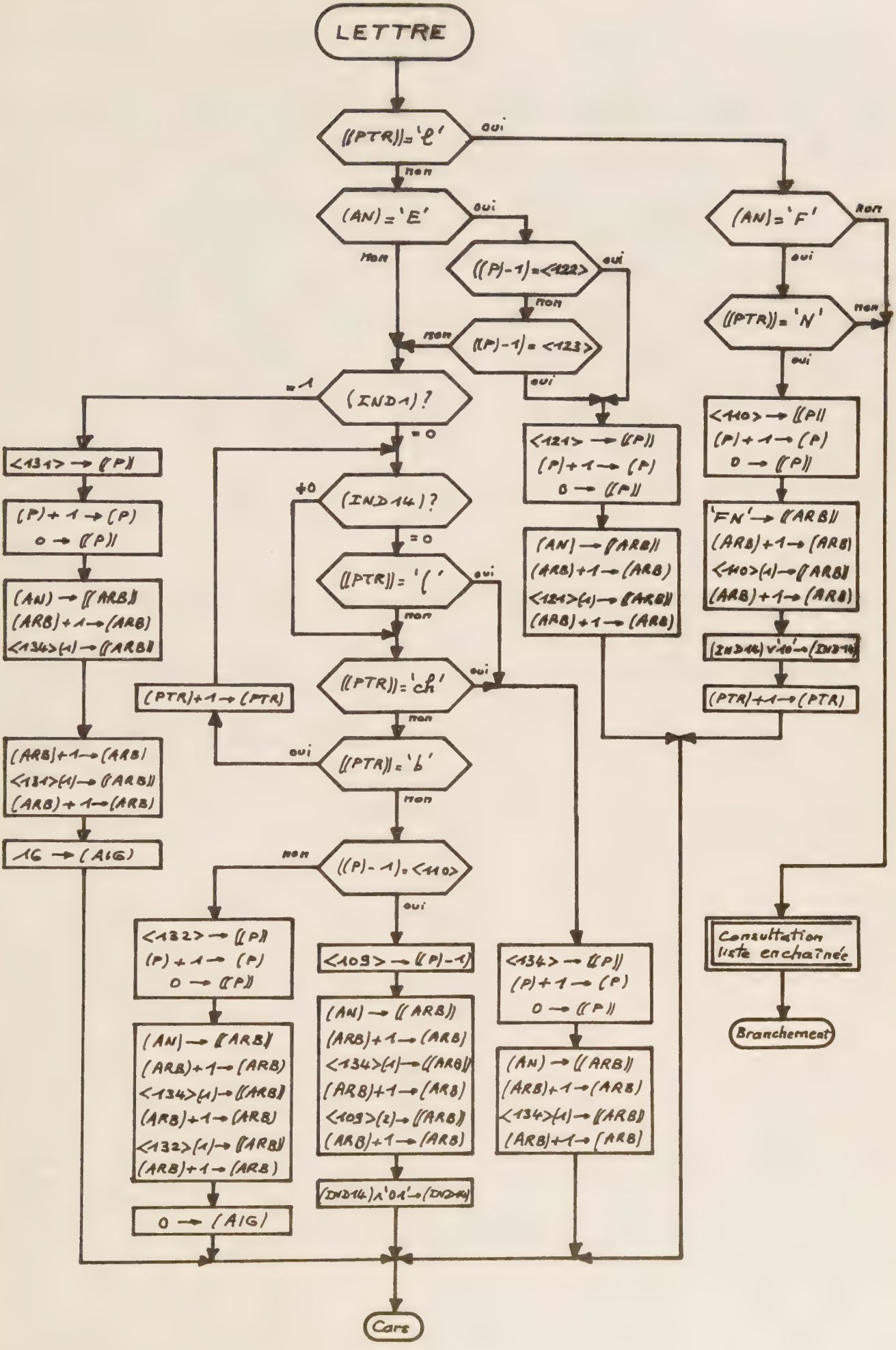

LETTRE
((PTR))='e'
oui
non
(AN)='E'
oui
non
((P)-1)=<122>
oui
non
((P)-1)=<123>
oui
non
(IND1)?
=1
=0
<131> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<134>(1) → ((ARB))
(ARB)+1 → (ARB)
(ARB)+1 → (ARB)
<131>(1) → ((ARB))
(ARB)+1 → (ARB)
16 → (AIG)
+0
(IND14)?
=0
((PTR))='('
oui
non
((PTR))='ch'
oui
non
(PTR)+1 → (PTR)
((PTR))='b'
oui
non
((P)-1)=<110>
non
oui
<132> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<134>(1) → ((ARB))
(ARB)+1 → (ARB)
<132>(1) → ((ARB))
(ARB)+1 → (ARB)
0 → (AIG)
<109> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<134>(1) → ((ARB))
(ARB)+1 → (ARB)
<109>(2) → ((ARB))
(ARB)+1 → (ARB)
(IND14)∧'01' → (IND14)
<121> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<121>(1) → ((ARB))
(ARB)+1 → (ARB)
<134> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<134>(1) → ((ARB))
(ARB)+1 → (ARB)
(AN)='F'
non
oui
((PTR))='N'
non
oui
<110> → ((P))
(P)+1 → (P)
0 → ((P))
'FN' → ((ARB))
(ARB)+1 → (ARB)
<110>(1) → ((ARB))
(ARB)+1 → (ARB)
(IND14) v '10' → (IND14)
(PTR)+1 → (PTR)
Consultation liste enchaînée
Branchement
Carte

Figure V-K

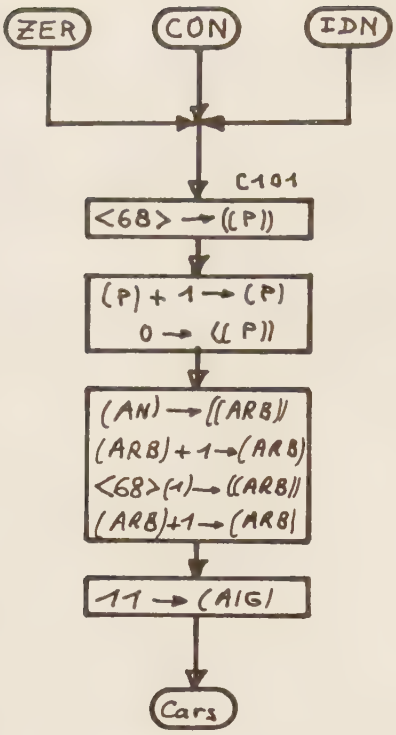


Figure V-11

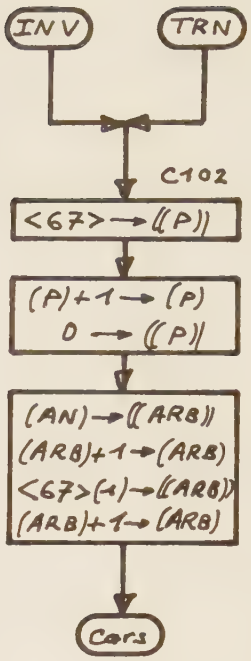


Figure V-12

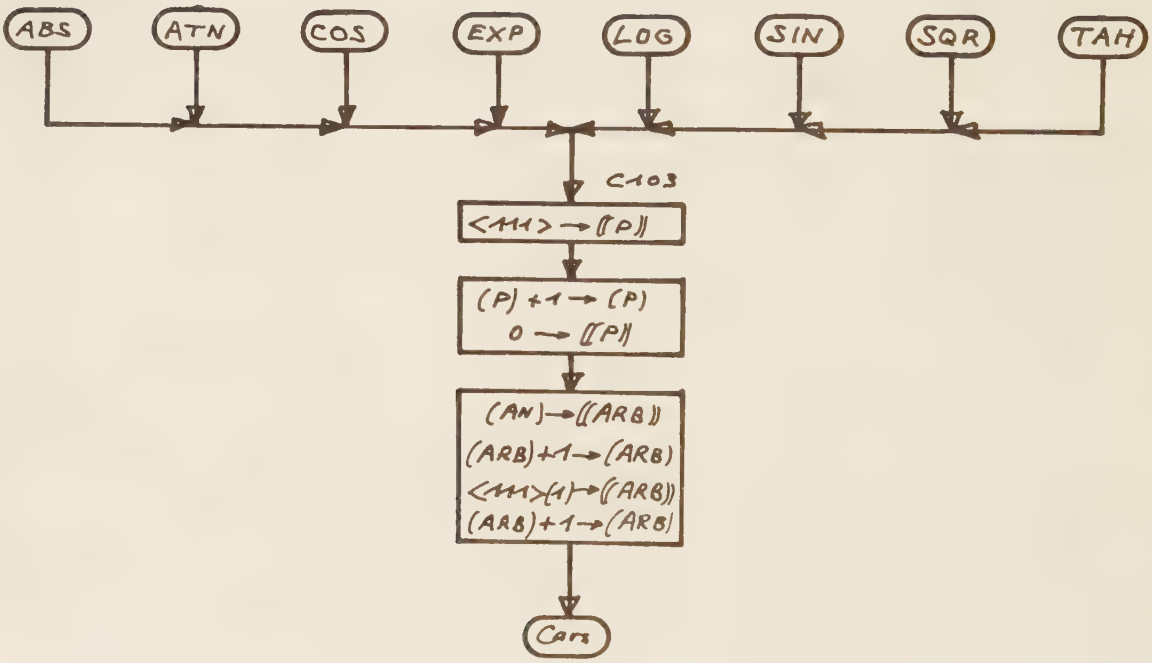


Figure V-13

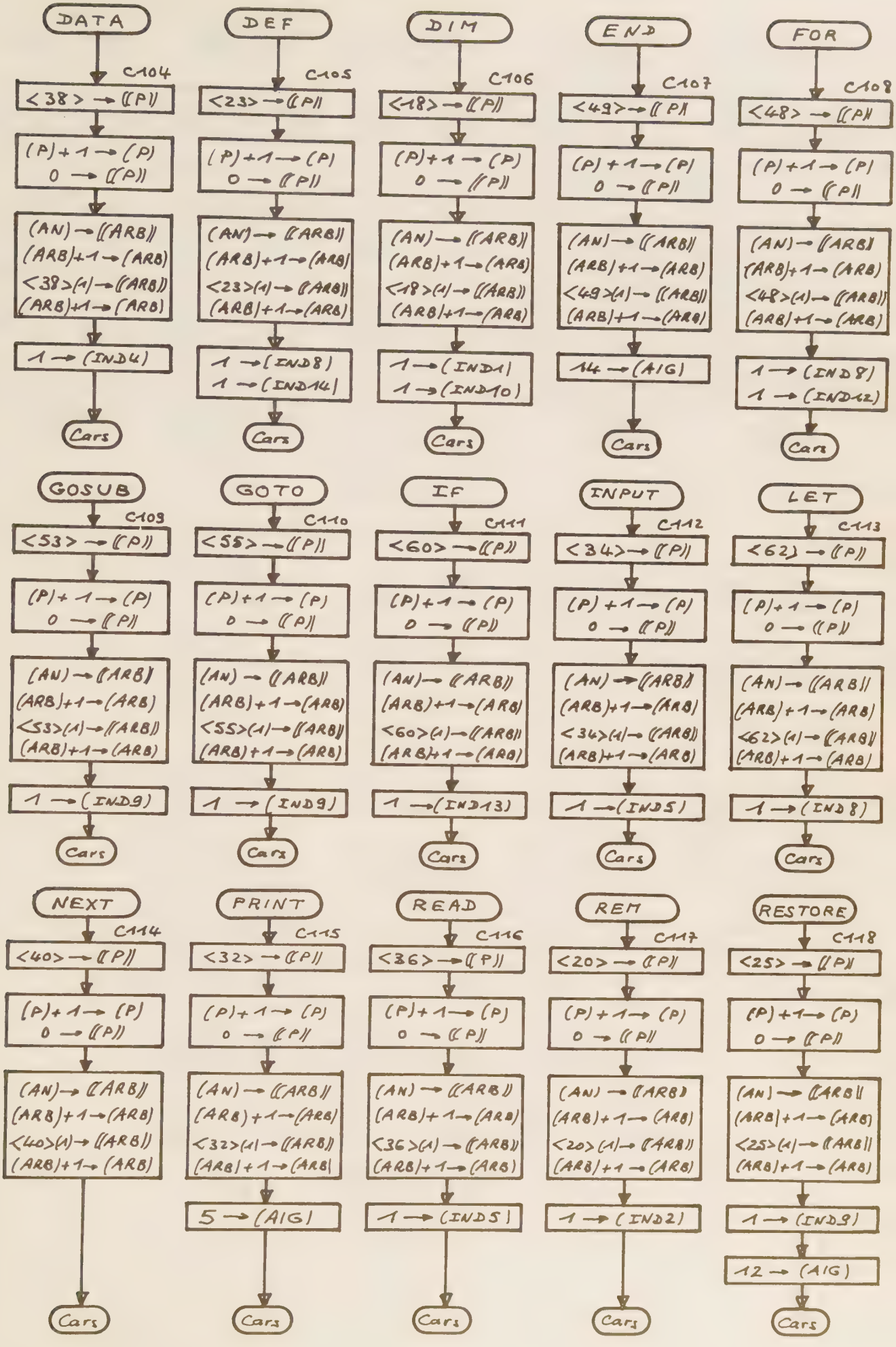

DATA
C104
<38> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<38>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND4)
Cars
DEF
C105
<23> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<23>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND8)
1 → (IND14)
Cars
DIM
C106
<18> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<18>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND1)
1 → (IND10)
Cars
END
C107
<49> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<49>(1) → ((ARB))
(ARB)+1 → (ARB)
14 → (AIG)
Cars
FOR
C108
<48> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<48>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND8)
1 → (IND12)
Cars
GOSUB
C109
<53> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<53>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND9)
Cars
GOTO
C110
<55> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<55>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND9)
Cars
IF
C111
<60> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<60>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND13)
Cars
INPUT
C112
<34> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<34>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND5)
Cars
LET
C113
<62> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<62>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND8)
Cars
NEXT
C114
<40> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<40>(1) → ((ARB))
(ARB)+1 → (ARB)
Cars
PRINT
C115
<32> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<32>(1) → ((ARB))
(ARB)+1 → (ARB)
5 → (AIG)
Cars
READ
C116
<36> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<36>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND5)
Cars
REM
C117
<20> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<20>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND2)
Cars
RESTORE
C118
<25> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<25>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND9)
12 → (AIG)
Cars

Figure V-24

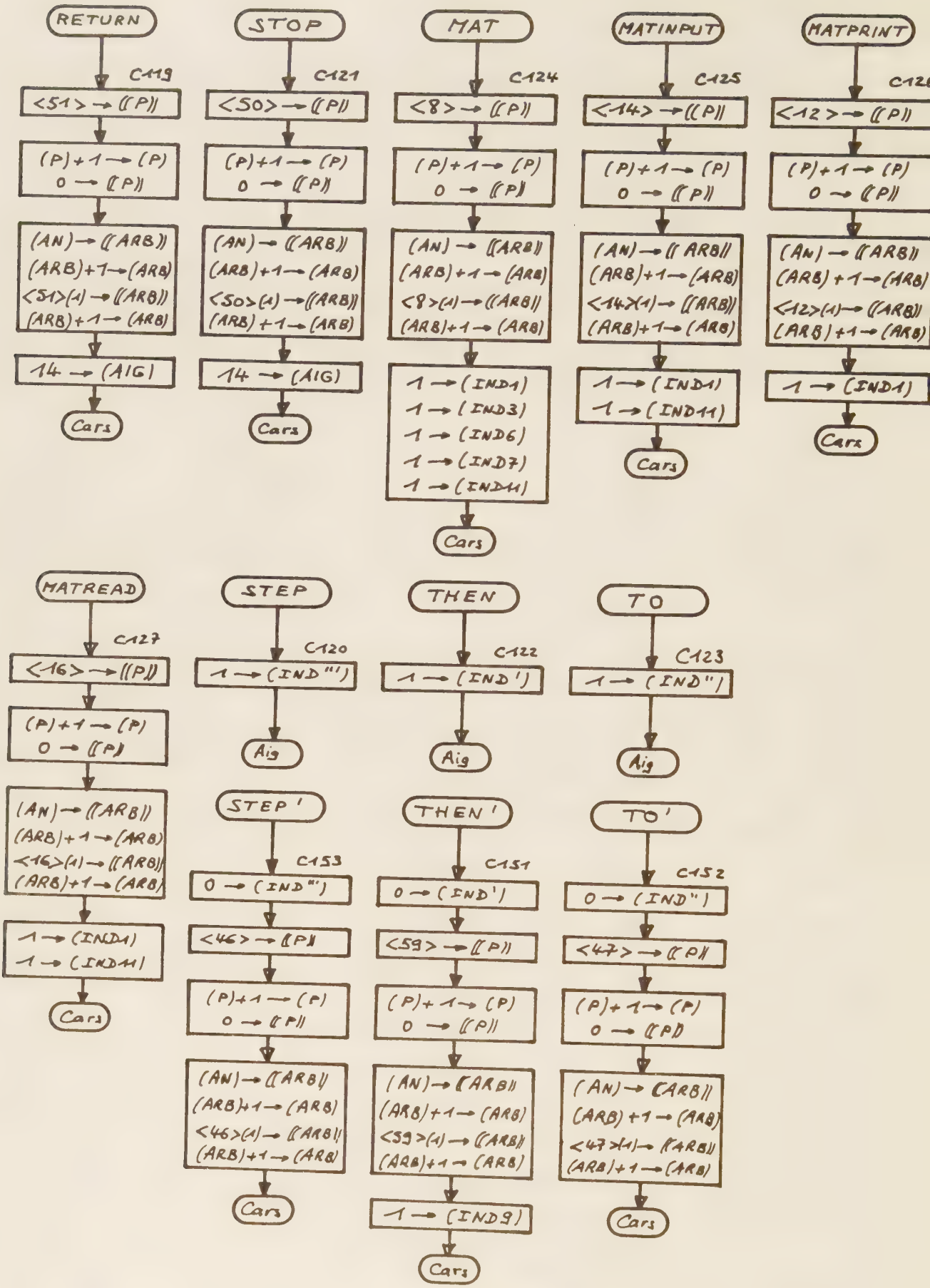

RETURN
C119
<51> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<51>(1) → ((ARB))
(ARB)+1 → (ARB)
14 → (AIG)
Cars
STOP
C121
<50> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<50>(1) → ((ARB))
(ARB)+1 → (ARB)
14 → (AIG)
Cars
MAT
C124
<8> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<8>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND1)
1 → (IND3)
1 → (IND6)
1 → (IND7)
1 → (IND11)
Cars
MATINPUT
C125
<14> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<14>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND1)
1 → (IND11)
Cars
MATPRINT
C12
<12> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<12>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND1)
Cars
MATREAD
C127
<16> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<16>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND1)
1 → (IND11)
Cars
STEP
C120
1 → (IND"')
Aig
STEP'
C153
0 → (IND"')
<46> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<46>(1) → ((ARB))
(ARB)+1 → (ARB)
Cars
THEN
C122
1 → (IND')
Aig
THEN'
C151
0 → (IND')
<59> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<59>(1) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND9)
Cars
TO
C123
1 → (IND")
Aig
TO'
C152
0 → (IND")
<47> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<47>(1) → ((ARB))
(ARB)+1 → (ARB)
Cars

Figure V-15

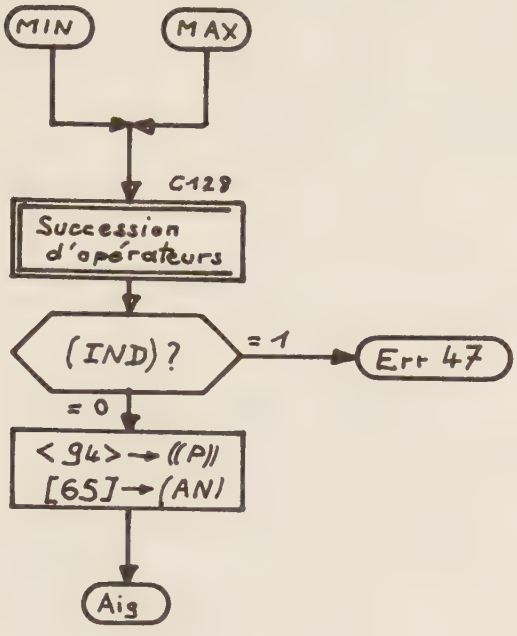


Figure V-16

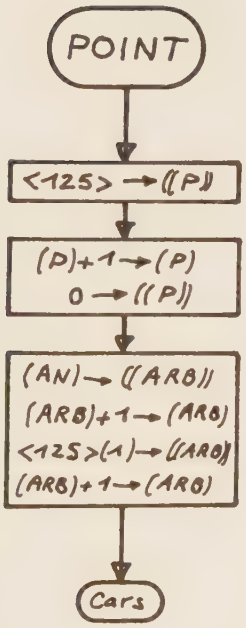


Figure V-17

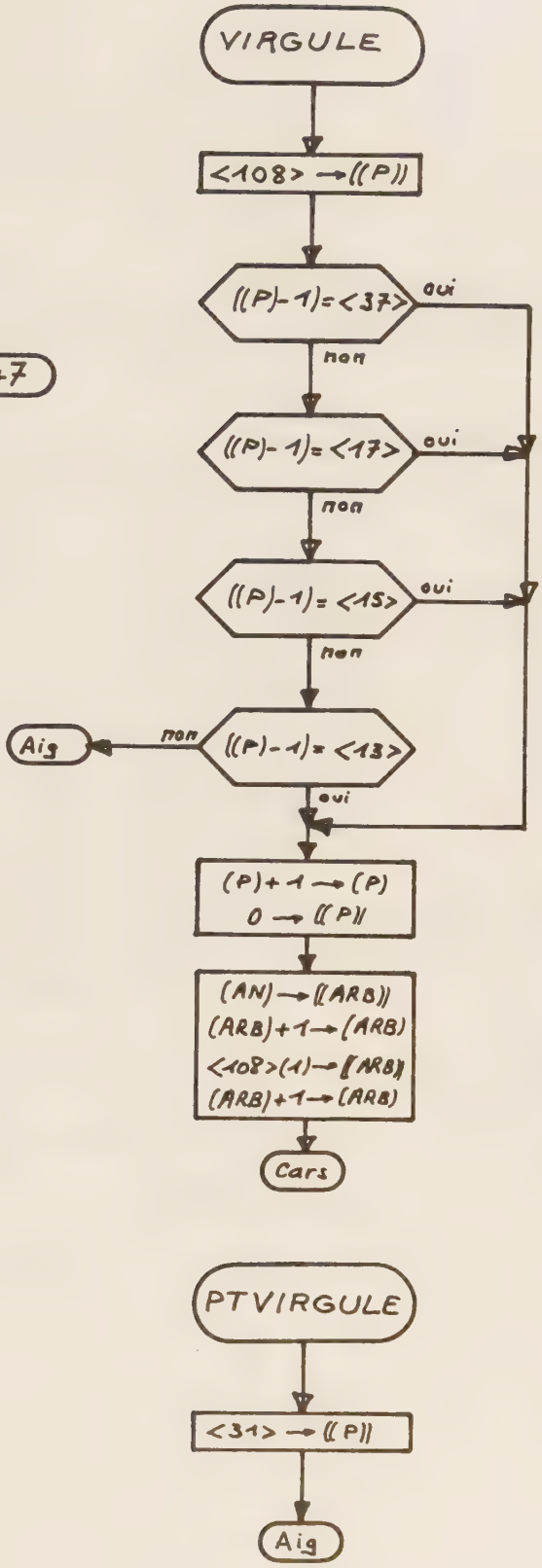


Figure V-18

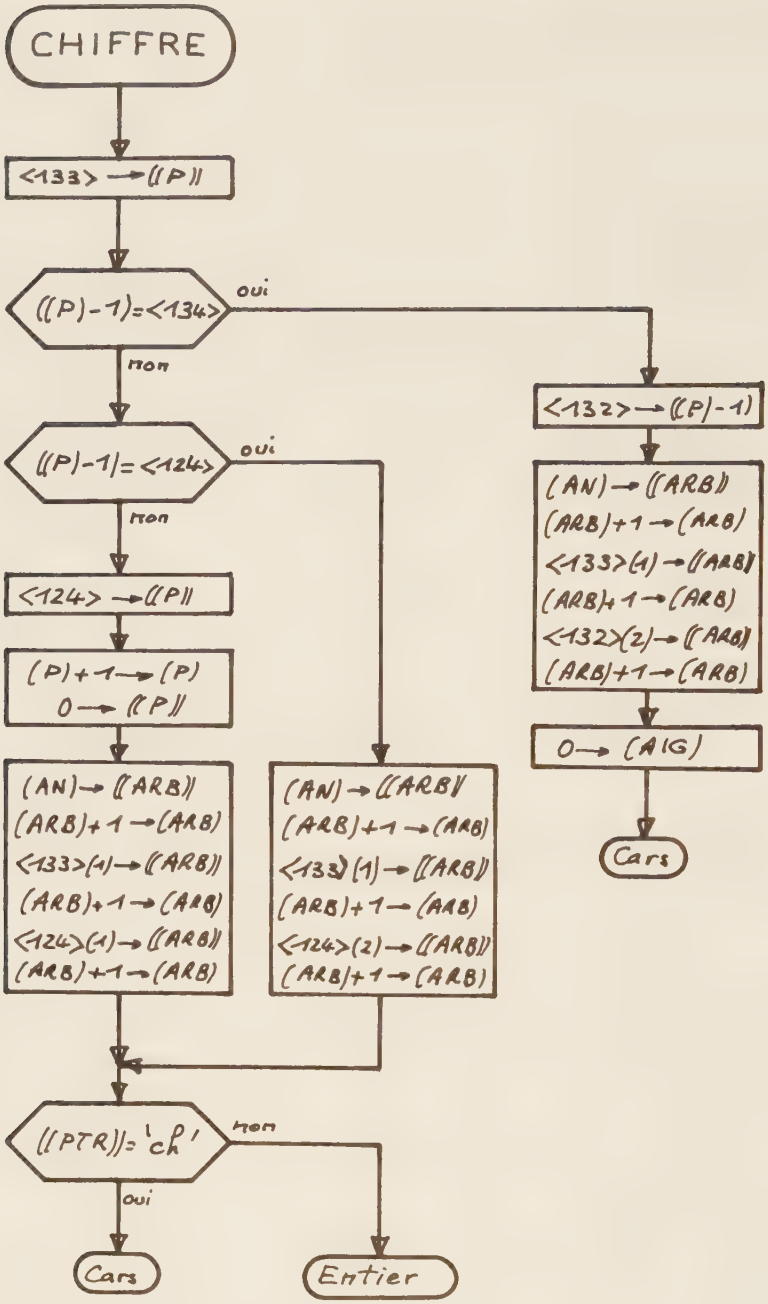

CHIFFRE
<133> → ((P))
((P)-1)=<134>
oui
non
<132> → ((P)-1)
((P)-1)=<124>
oui
non
(AN) → ((ARB))
(ARB)+1 → (ARB)
<133>(1) → ((ARB))
(ARB)+1 → (ARB)
<132>(2) → ((ARB))
(ARB)+1 → (ARB)
0 → (AIG)
Cars
<124> → ((P))
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<133>(1) → ((ARB))
(ARB)+1 → (ARB)
<124>(1) → ((ARB))
(ARB)+1 → (ARB)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<133>(1) → ((ARB))
(ARB)+1 → (ARB)
<124>(2) → ((ARB))
(ARB)+1 → (ARB)
((PTR))='ch'
non
oui
Cars
Entier

Figure V-19

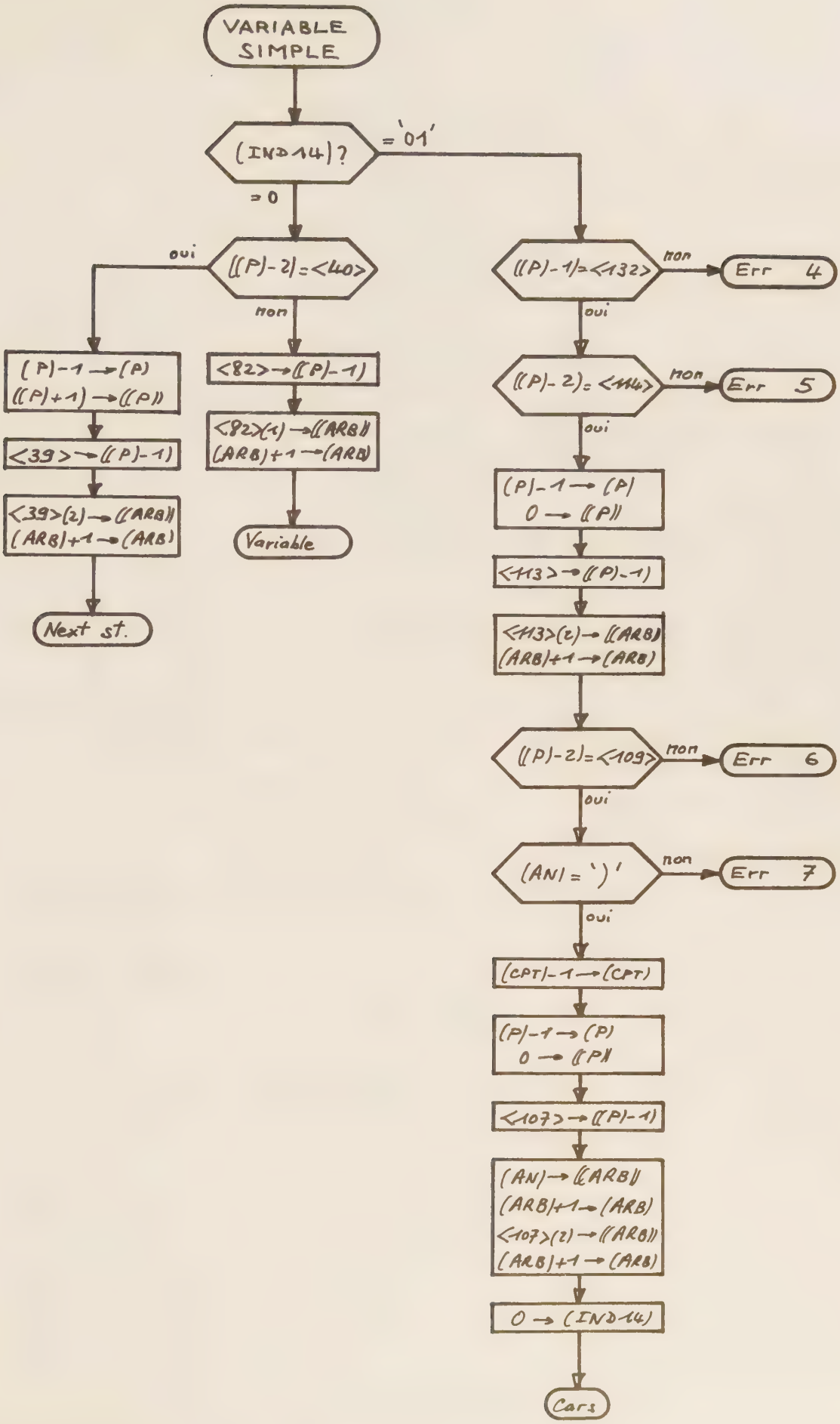

VARIABLE SIMPLE
(IND14)?
='01'
=0
oui
((P)-2)=<40>
non
(P)-1→(P)
((P)+1)→((P))
<39>→((P)-1)
<39>(2)→((ARB))
(ARB)+1→(ARB)
Next st.
<82>→((P)-1)
<82>(1)→((ARB))
(ARB)+1→(ARB)
Variable
((P)-1)=<132>
non
Err 4
oui
((P)-2)=<114>
non
Err 5
oui
(P)-1→(P)
0→((P))
<113>→((P)-1)
<113>(2)→((ARB))
(ARB)+1→(ARB)
((P)-2)=<109>
non
Err 6
oui
(AN)=')'
non
Err 7
oui
(CPT)-1→(CPT)
(P)-1→(P)
0→((P))
<107>→((P)-1)
(AN)→((ARB))
(ARB)+1→(ARB)
<107>(2)→((ARB))
(ARB)+1→(ARB)
0→(IND14)
Cars

Figure V-20

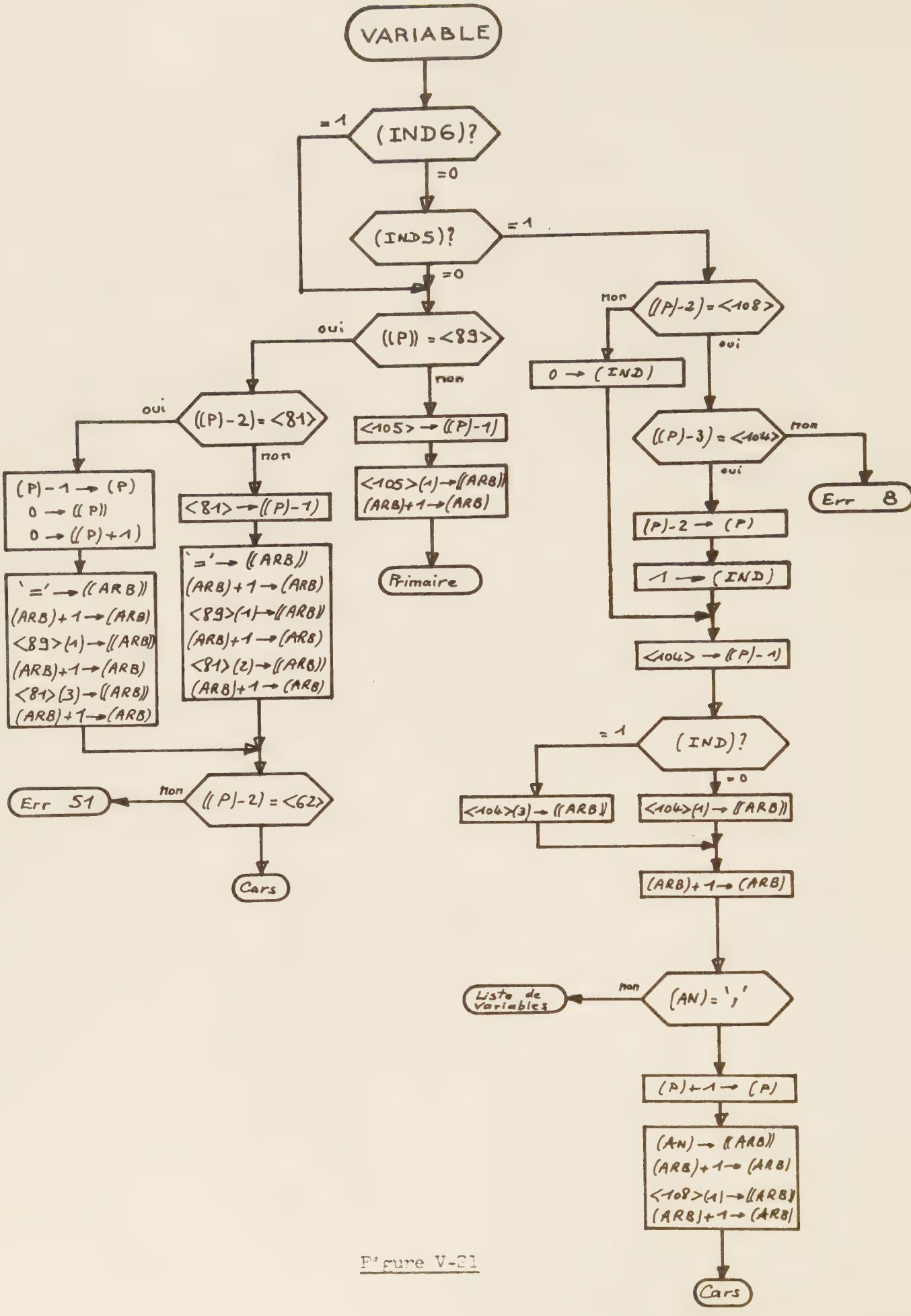

VARIABLE
(IND6)?
=1
=0
(IND5)?
=1
=0
((P)) =<89>
oui
non
((P)-2)=<81>
oui
non
(P)-1 → (P)
0 → ((P))
0 → ((P)+1)
'=' → ((ARB))
(ARB)+1 → (ARB)
<89>(1) → ((ARB))
(ARB)+1 → (ARB)
<81>(3) → ((ARB))
(ARB)+1 → (ARB)
<81> → ((P)-1)
'=' → ((ARB))
(ARB)+1 → (ARB)
<89>(1) → ((ARB))
(ARB)+1 → (ARB)
<81>(2) → ((ARB))
(ARB)+1 → (ARB)
<105> → ((P)-1)
<105>(1) → ((ARB))
(ARB)+1 → (ARB)
Primaire
((P)-2) = <62>
non
Err 51
Cars
((P)-2) = <108>
non
oui
0 → (IND)
((P)-3) = <104>
non
oui
Err 8
(P)-2 → (P)
1 → (IND)
<104> → ((P)-1)
(IND)?
=1
=0
<104>(3) → ((ARB))
<104>(1) → ((ARB))
(ARB)+1 → (ARB)
(AN) = ','
non
Liste de Variables
(P)+1 → (P)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<108>(1) → ((ARB))
(ARB)+1 → (ARB)
Cars

Figure V-21

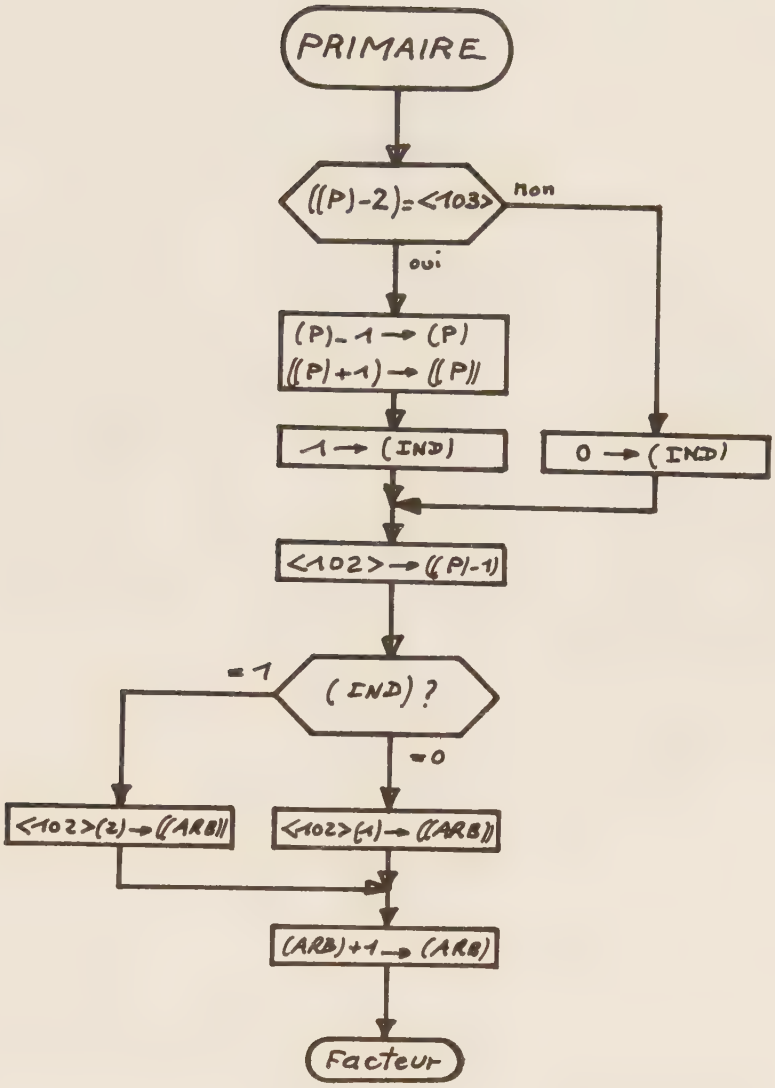

PRIMAIRE
((P)-2)=<103>
non
oui
(P)-1 → (P)
((P)+1) → ((P))
1 → (IND)
0 → (IND)
<102> → ((P)-1)
(IND) ?
=1
=0
<102>(2) → ((ARB))
<102>(1) → ((ARB))
(ARB)+1 → (ARB)
Facteur

Figure V-22

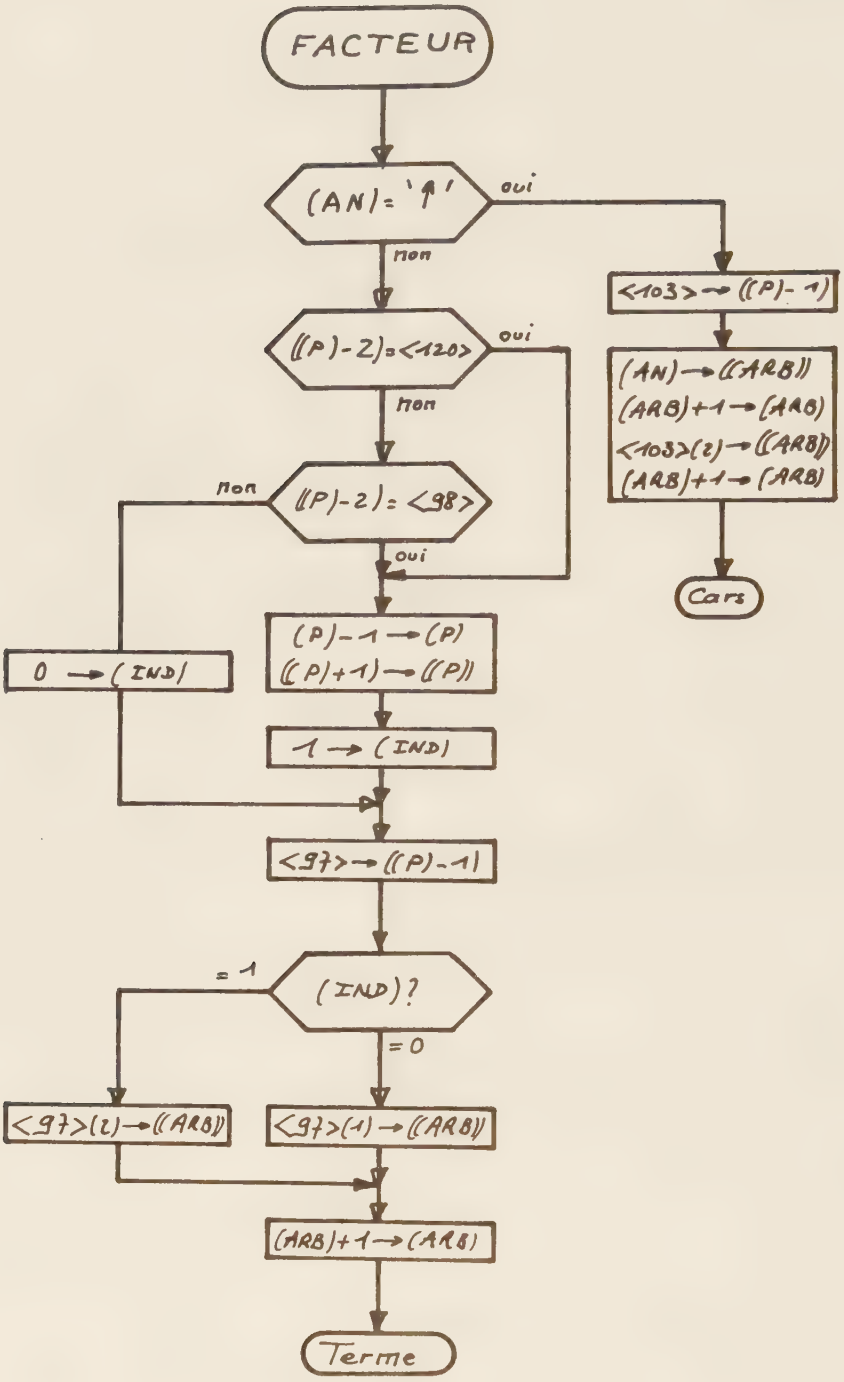

FACTEUR
(AN)='↑'
oui
non
<103>→((P)-1)
((P)-2)=<120>
oui
non
(AN)→((ARB))
(ARB)+1→(ARB)
<103>(2)→((ARB))
(ARB)+1→(ARB)
non
((P)-2)=<98>
oui
Cars
(P)-1→(P)
((P)+1)→((P))
0→(IND)
1→(IND)
<97>→((P)-1)
=1
(IND)?
=0
<97>(2)→((ARB))
<97>(1)→((ARB))
(ARB)+1→(ARB)
Terme

Figure V-23

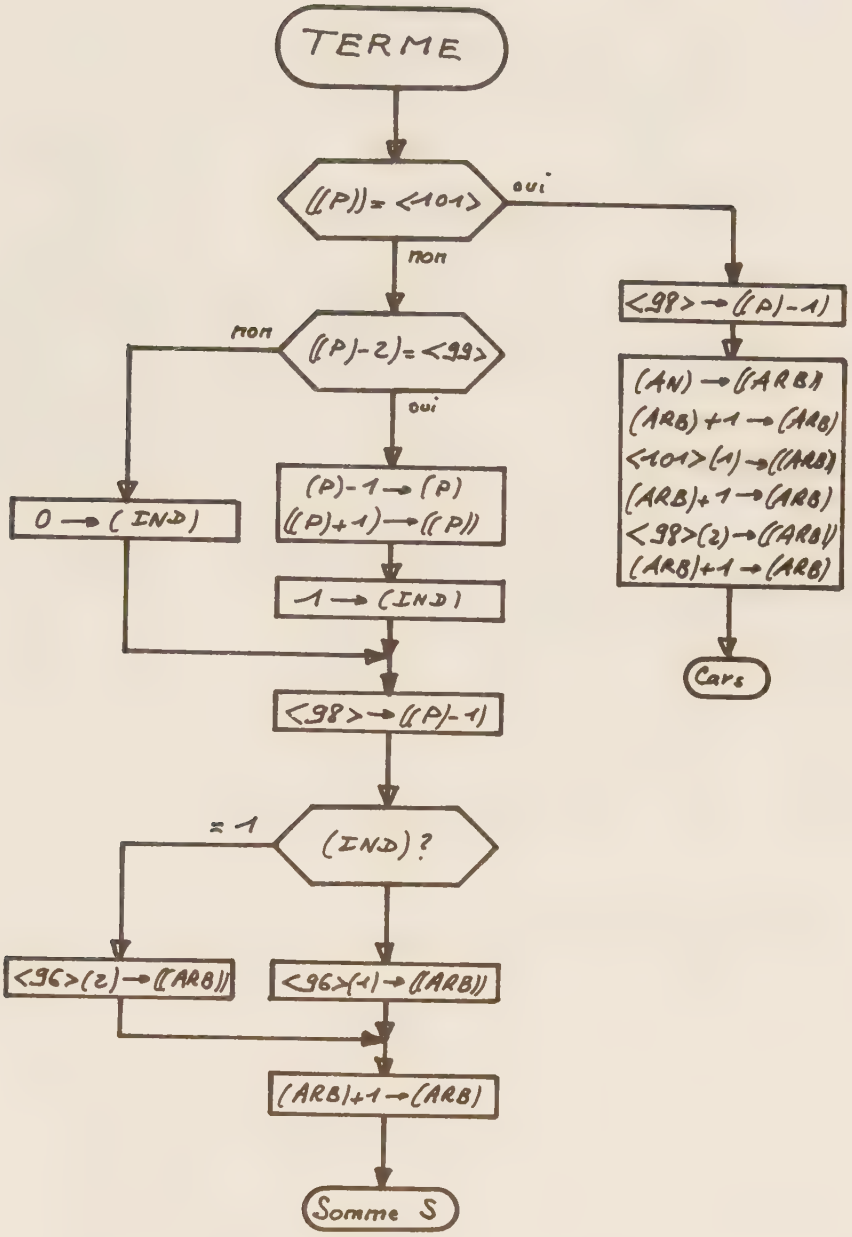

TERME
((P)) = <101>
oui
non
<98> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<101>(-1) → ((ARB))
(ARB)+1 → (ARB)
<98>(2) → ((ARB))
(ARB)+1 → (ARB)
Cars
non
((P)-2) = <99>
oui
0 → (IND)
(P)-1 → (P)
((P)+1) → ((P))
-1 → (IND)
<98> → ((P)-1)
= 1
(IND) ?
<96>(2) → ((ARB))
<96>(-1) → ((ARB))
(ARB)+1 → (ARB)
Somme S

Figure V-24

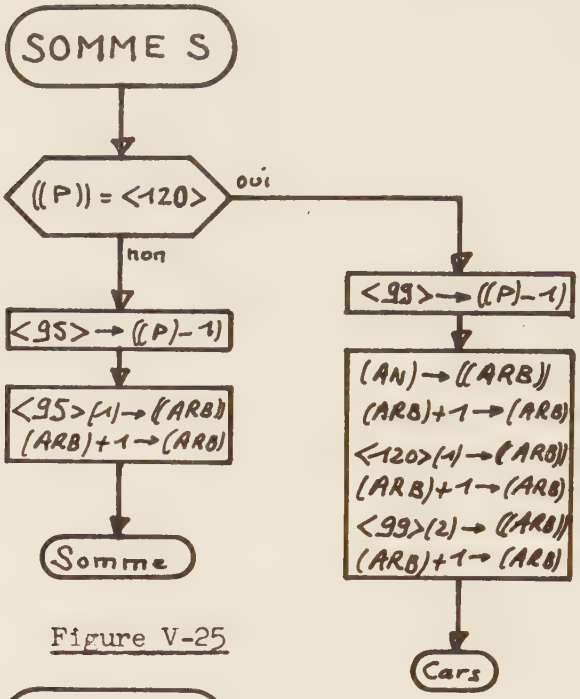


Figure V-25

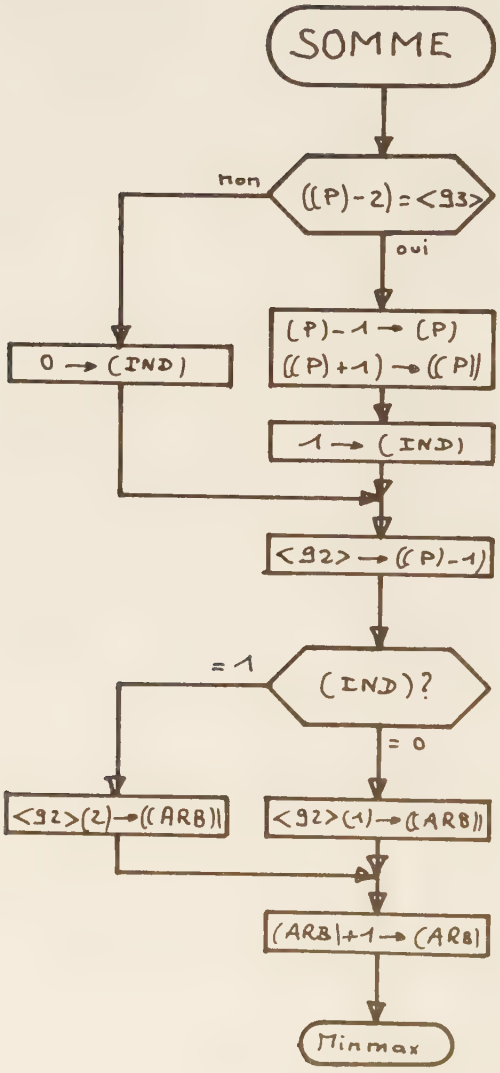


Figure V-26

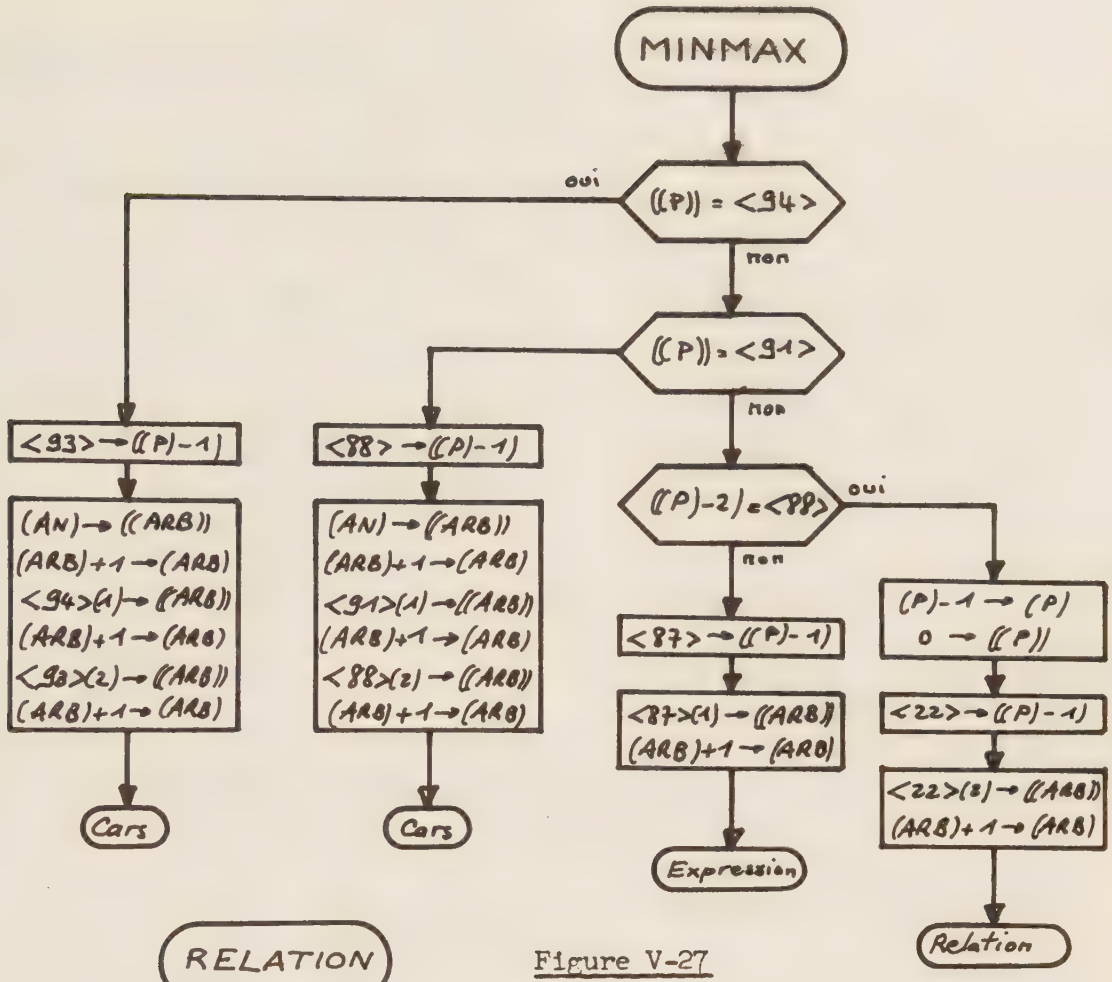


Figure V-27

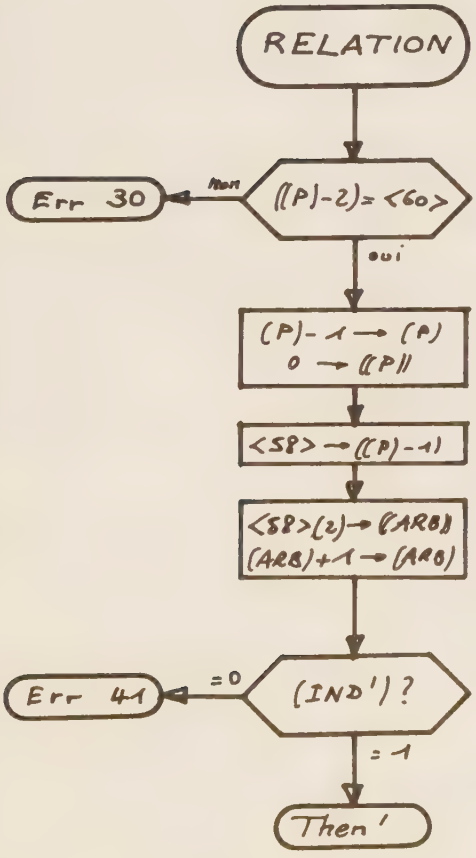


Figure V-28

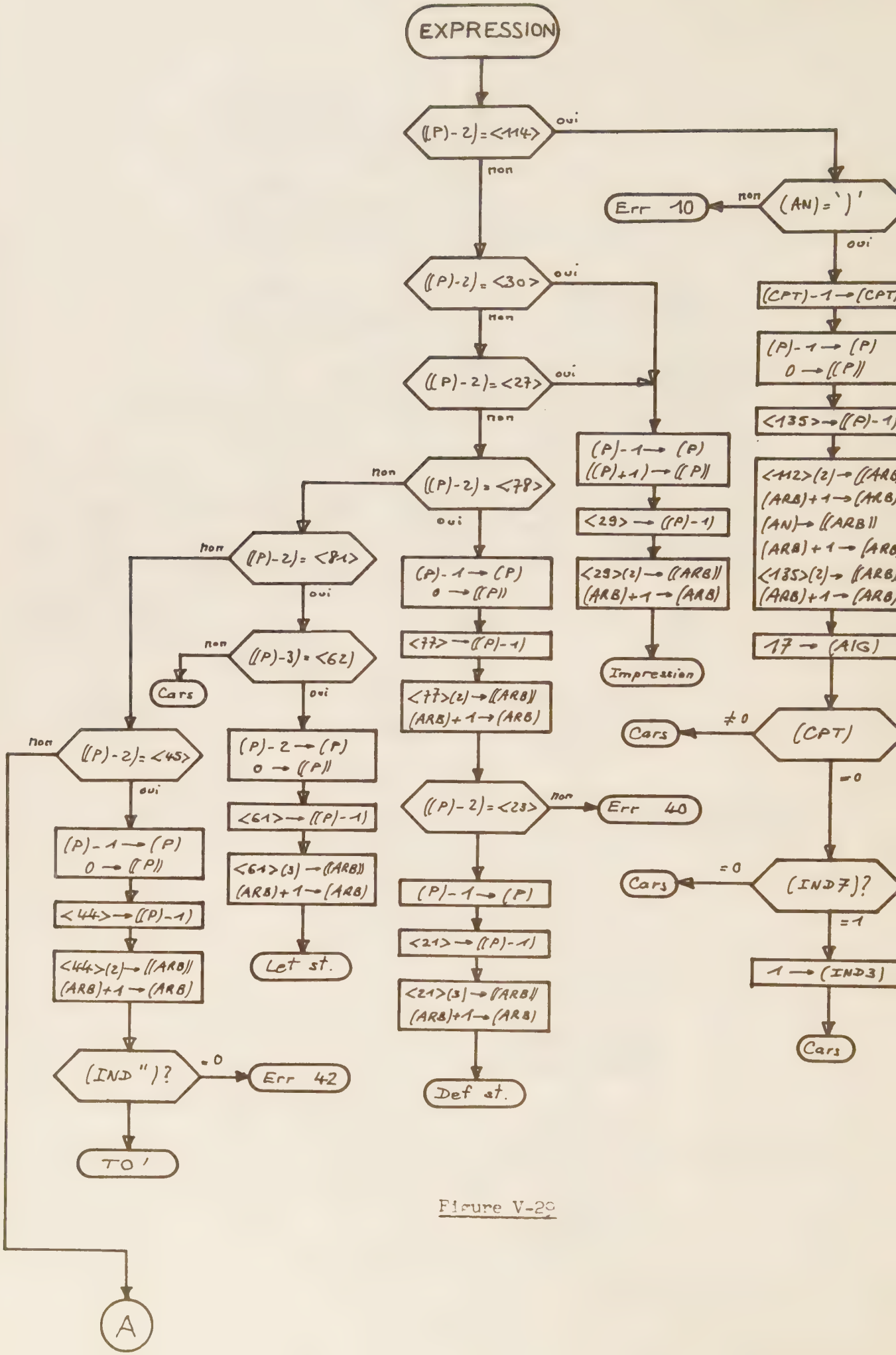

EXPRESSION
((P)-2)=<114>
oui
non
(AN)=')'
non
Err 10
oui
(CPT)-1→(CPT)
((P)-2)=<30>
oui
non
((P)-2)=<27>
oui
non
(P)-1→(P)
0→((P))
<135>→((P)-1)
(P)-1→(P)
((P)+1)→((P))
<29>→((P)-1)
<29>(2)→((ARB))
(ARB)+1→(ARB)
Impression
<112>(2)→((ARB
(ARB)+1→(ARB)
(AN)→((ARB))
(ARB)+1→(ARB)
<135>(2)→((ARB
(ARB)+1→(ARB)
17→(AIG)
((P)-2)=<78>
non
oui
((P)-2)=<81>
non
oui
((P)-3)=<62>
non
oui
Cars
(P)-1→(P)
0→((P))
<77>→((P)-1)
<77>(2)→((ARB))
(ARB)+1→(ARB)
((P)-2)=<23>
non
Err 40
(P)-1→(P)
<21>→((P)-1)
<21>(3)→((ARB))
(ARB)+1→(ARB)
Def st.
(P)-2→(P)
0→((P))
<61>→((P)-1)
<61>(3)→((ARB))
(ARB)+1→(ARB)
Let st.
Cars
≠0
(CPT)
=0
Cars
=0
(IND7)?
=1
1→(IND3)
Cars
((P)-2)=<45>
non
oui
(P)-1→(P)
0→((P))
<44>→((P)-1)
<44>(2)→((ARB))
(ARB)+1→(ARB)
(IND")?
=0
Err 42
TO'
A

Figure V-29

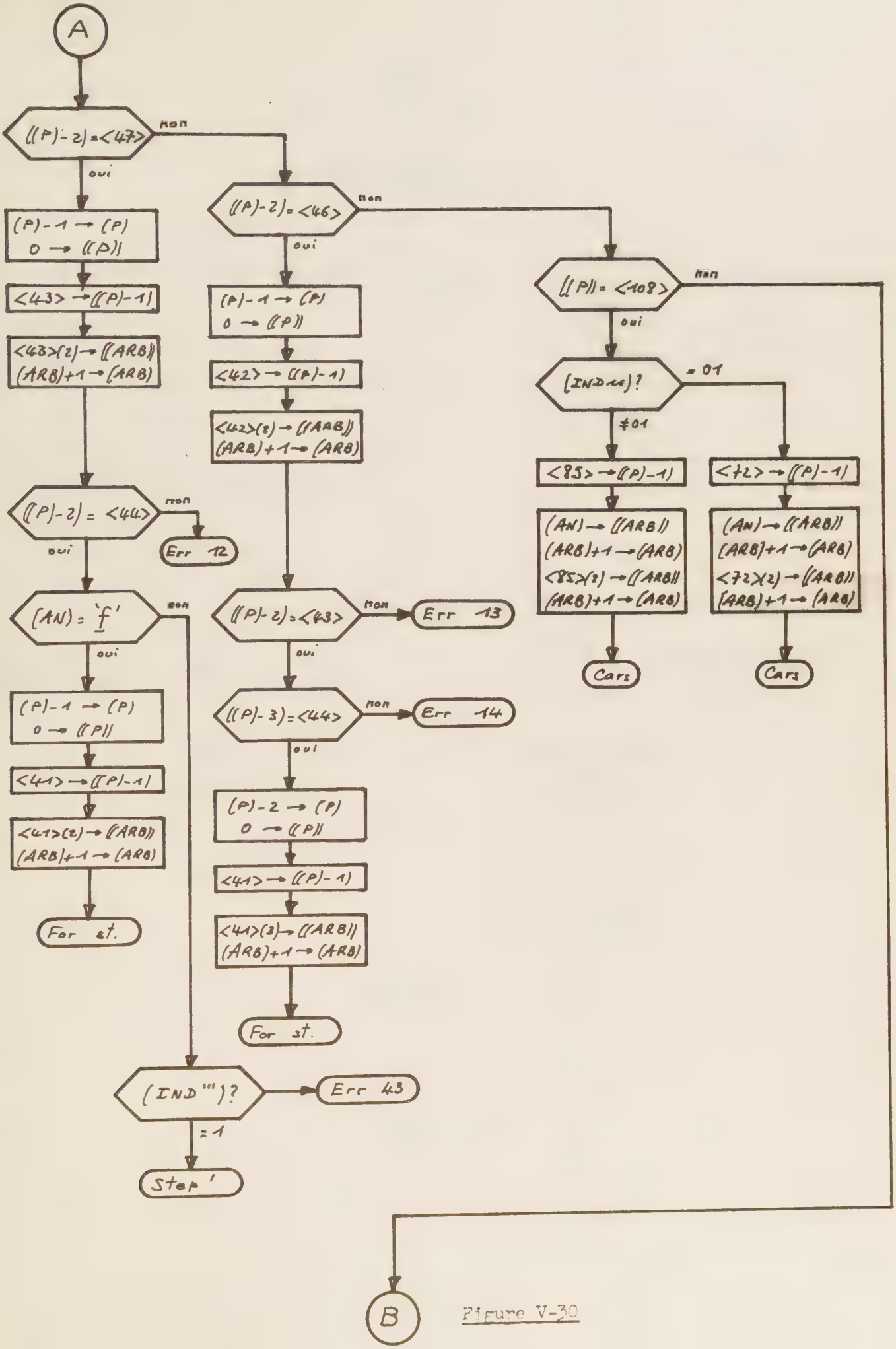

A
((P)-2)=<47>
non
oui
(P)-1 → (P)
0 → ((P))
<43> → ((P)-1)
<43>(2) → ((ARB))
(ARB)+1 → (ARB)
((P)-2) = <44>
non
Err 12
oui
(AN) = 'f'
non
oui
(P)-1 → (P)
0 → ((P))
<41> → ((P)-1)
<41>(2) → ((ARB))
(ARB)+1 → (ARB)
For st.
(IND''')?
Err 43
=1
Step'
((P)-2)=<46>
non
oui
(P)-1 → (P)
0 → ((P))
<42> → ((P)-1)
<42>(2) → ((ARB))
(ARB)+1 → (ARB)
((P)-2)=<43>
non
Err 13
oui
((P)-3)=<44>
non
Err 14
oui
(P)-2 → (P)
0 → ((P))
<41> → ((P)-1)
<41>(3) → ((ARB))
(ARB)+1 → (ARB)
For st.
((P))=<108>
non
oui
(IND'')?
=01
≠01
<85> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<85>(2) → ((ARB))
(ARB)+1 → (ARB)
Cars
<72> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<72>(2) → ((ARB))
(ARB)+1 → (ARB)
Cars
B

Figure V-30

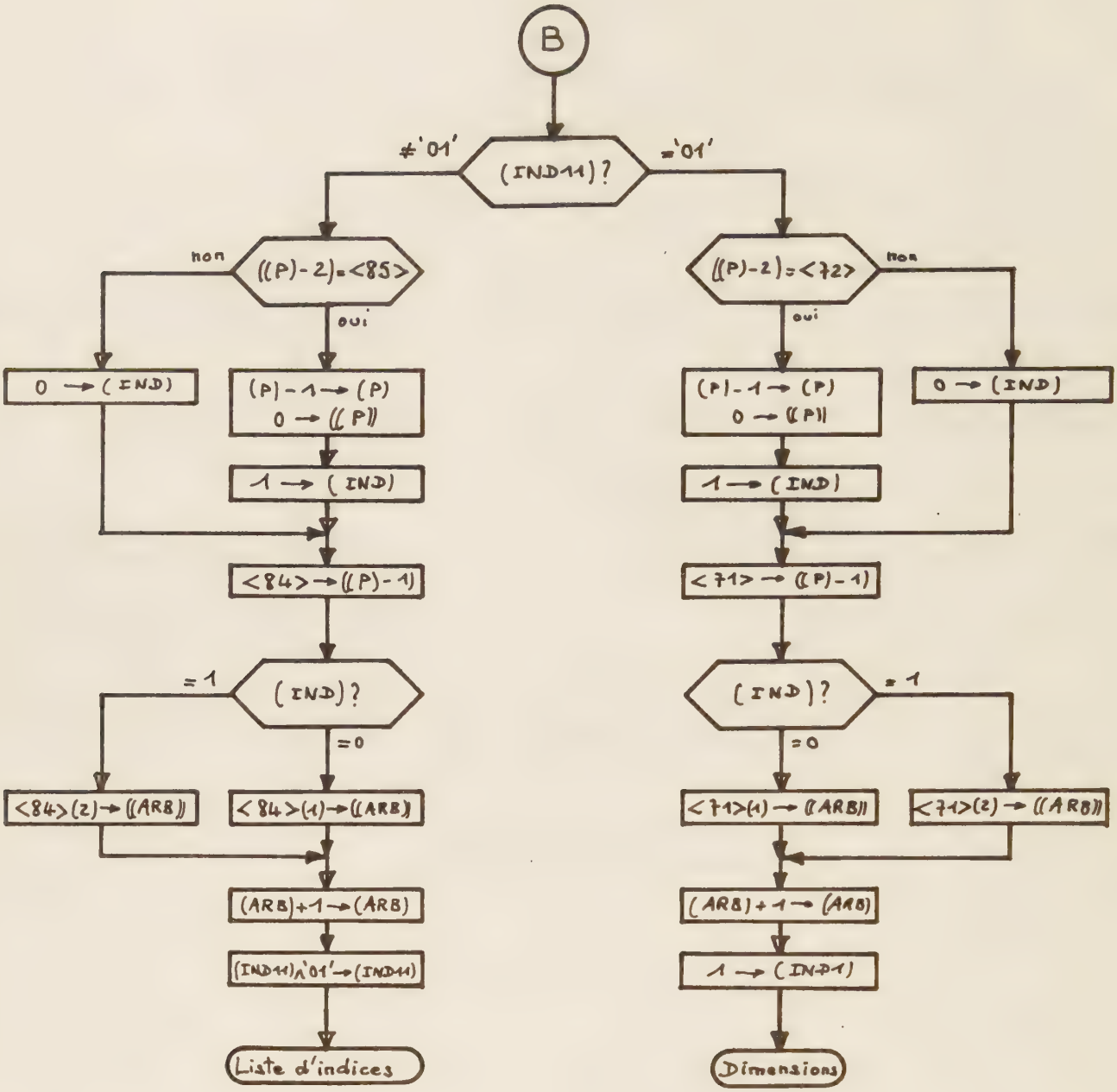

B
(IND11) ?
≠'01'
='01'
non
((P)-2)=<85>
oui
((P)-2)=<72>
non
oui
0 → (IND)
(P)-1 → (P)
0 → ((P))
1 → (IND)
<84> → ((P)-1)
(IND) ?
=1
=0
<84>(2) → ((ARB))
<84>(1) → ((ARB))
(ARB)+1 → (ARB)
(IND11)∧'01' → (IND11)
Liste d'indices
(P)-1 → (P)
0 → ((P))
0 → (IND)
1 → (IND)
<71> → ((P)-1)
(IND) ?
=1
=0
<71>(1) → ((ARB))
<71>(2) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND1)
Dimensions

Figure V-31

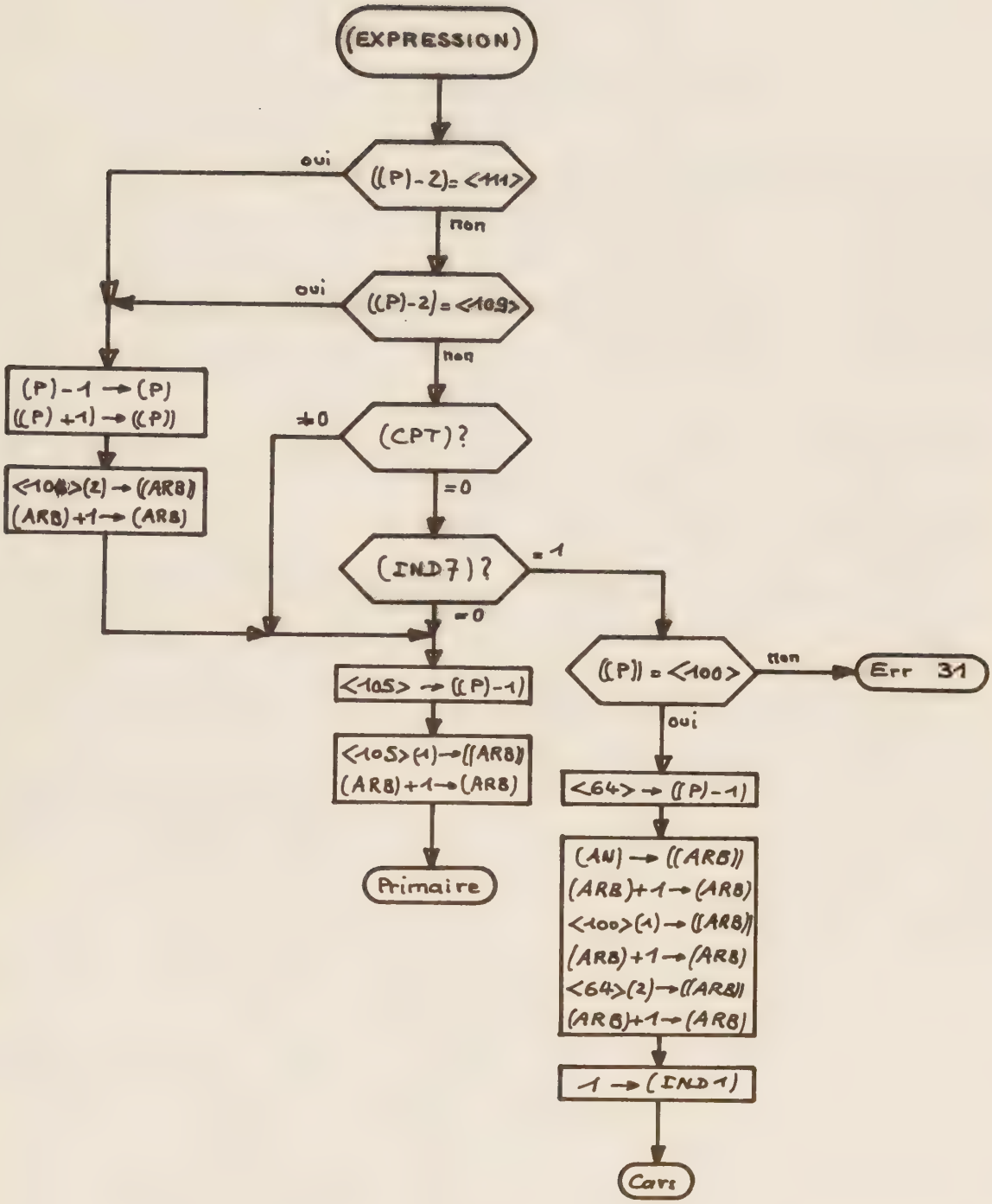

(EXPRESSION)
((P)-2)= <111>
oui
non
((P)-2) = <109>
oui
non
(P)-1 → (P)
((P)+1) → ((P))
<10$>(2) → ((ARB))
(ARB)+1 → (ARB)
(CPT)?
≠0
=0
(IND7)?
=1
=0
<105> → ((P)-1)
<105>(1) → ((ARB))
(ARB)+1 → (ARB)
Primaire
((P)) = <100>
non
Err 31
oui
<64> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<100>(1) → ((ARB))
(ARB)+1 → (ARB)
<64>(2) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND1)
Cart

Figure V-32

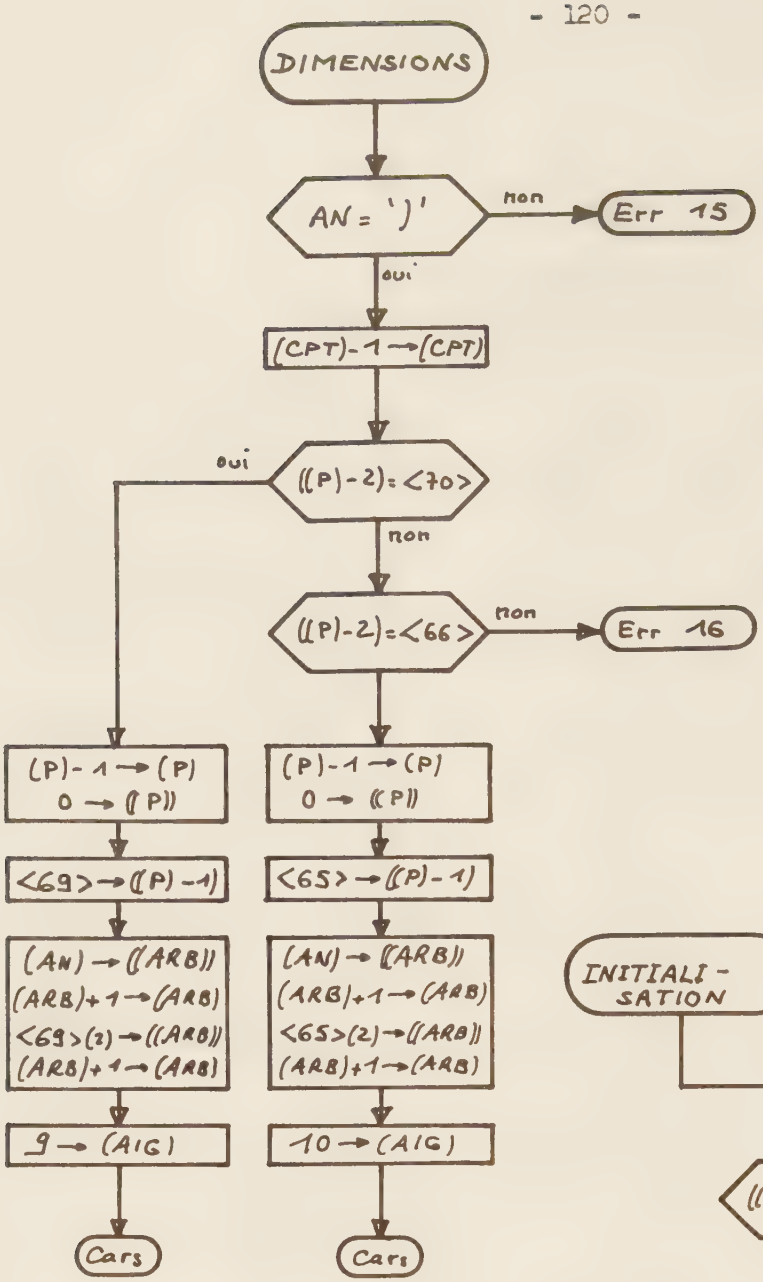


Figure V-33

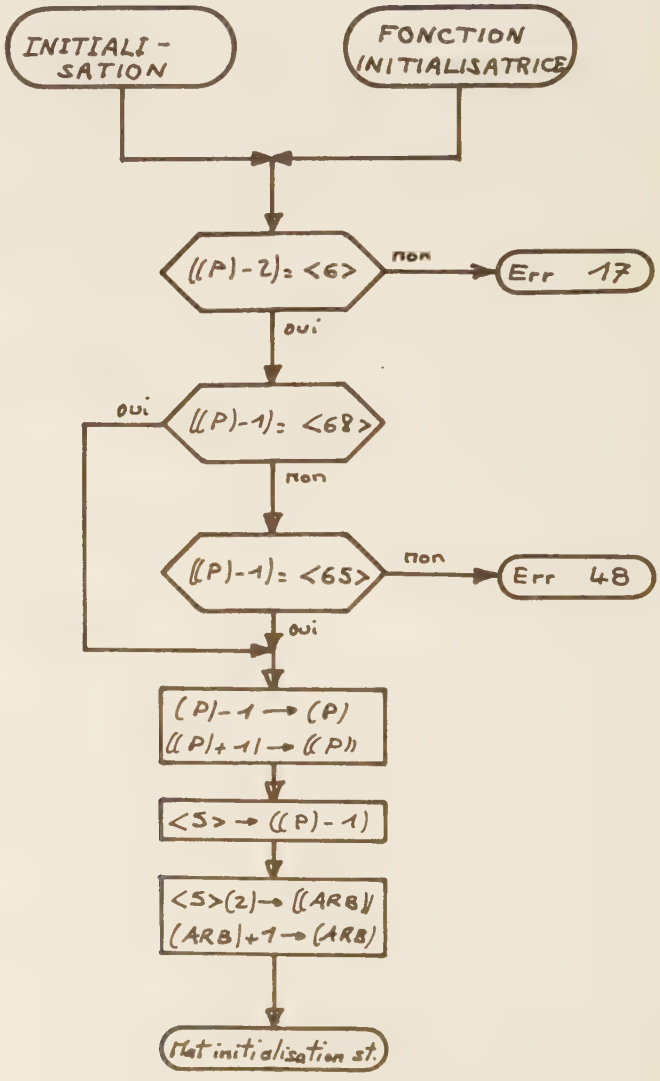


Figure V-34

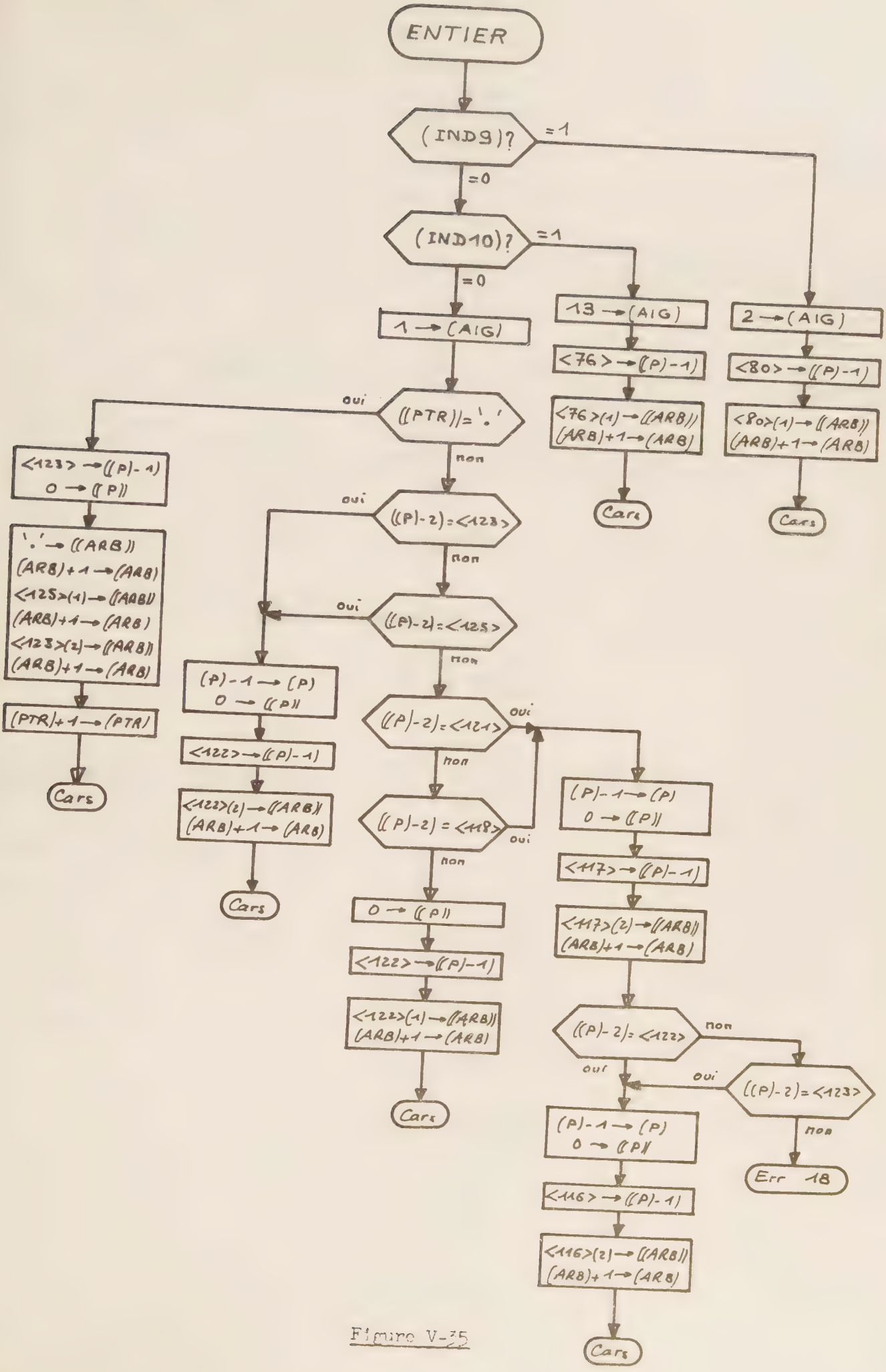

ENTIER
(IND9)?
=1
=0
(IND10)?
=1
=0
13 → (AIG)
2 → (AIG)
1 → (AIG)
<76> → ((P)-1)
<80> → ((P)-1)
<76>(1) → ((ARB))
(ARB)+1 → (ARB)
<80>(1) → ((ARB))
(ARB)+1 → (ARB)
Cars
Cars
((PTR))='.'
oui
non
<123> → ((P)-1)
0 → ((P))
'.' → ((ARB))
(ARB)+1 → (ARB)
<125>(1) → ((ARB))
(ARB)+1 → (ARB)
<123>(2) → ((ARB))
(ARB)+1 → (ARB)
(PTR)+1 → (PTR)
Cars
((P)-2)=<123>
oui
non
(P)-1 → (P)
0 → ((P))
<122> → ((P)-1)
<122>(2) → ((ARB))
(ARB)+1 → (ARB)
Cars
((P)-2)=<125>
oui
non
((P)-2)=<121>
oui
non
((P)-2)=<118>
oui
non
0 → ((P))
<122> → ((P)-1)
<122>(1) → ((ARB))
(ARB)+1 → (ARB)
Cars
(P)-1 → (P)
0 → ((P))
<117> → ((P)-1)
<117>(2) → ((ARB))
(ARB)+1 → (ARB)
((P)-2)=<122>
non
oui
oui
((P)-2)=<123>
non
(P)-1 → (P)
0 → ((P))
<116> → ((P)-1)
<116>(2) → ((ARB))
(ARB)+1 → (ARB)
Err 18
Cars

Figure V-25

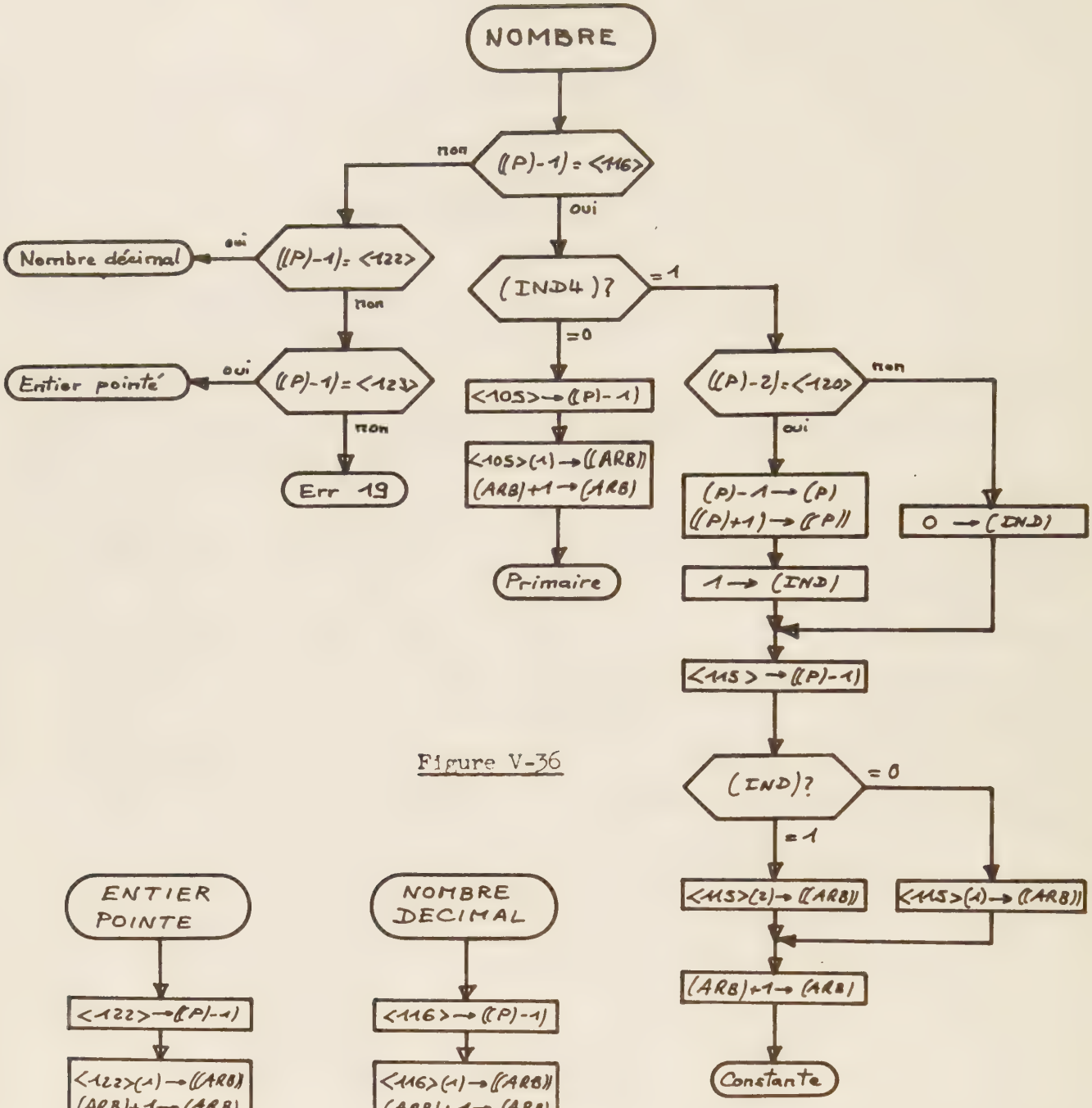


Figure V-36

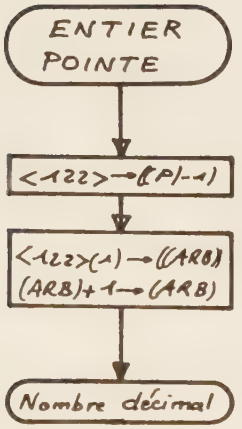


Figure V-37

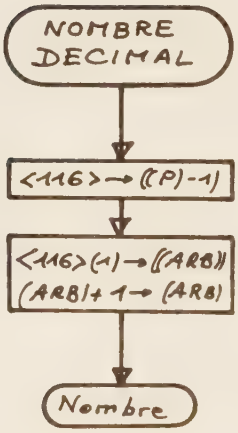


Figure V-38

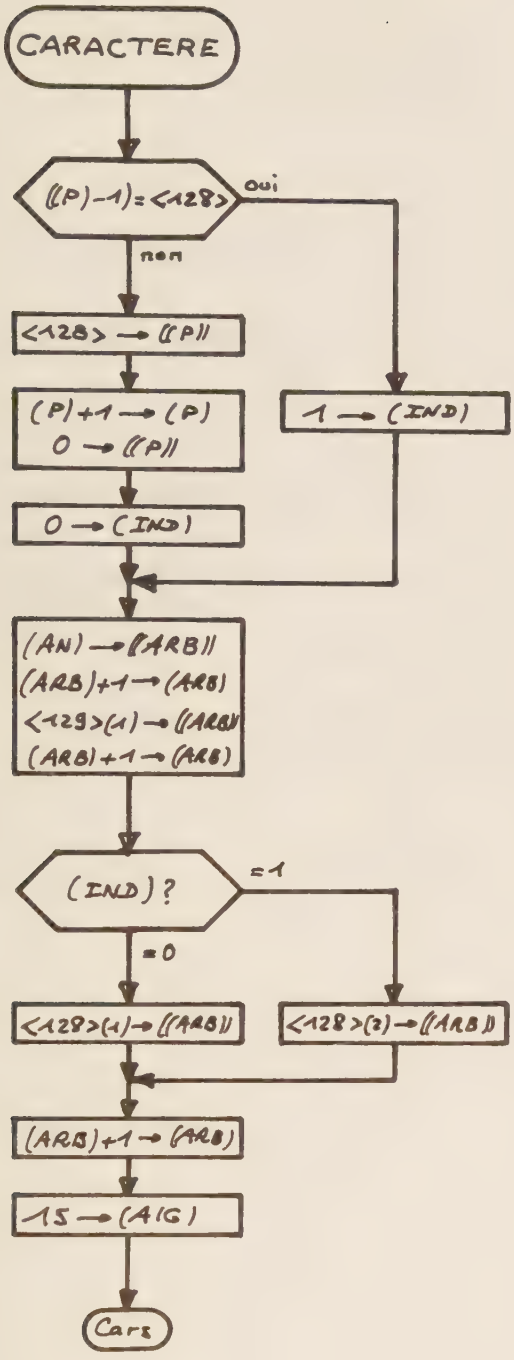


Figure V-39

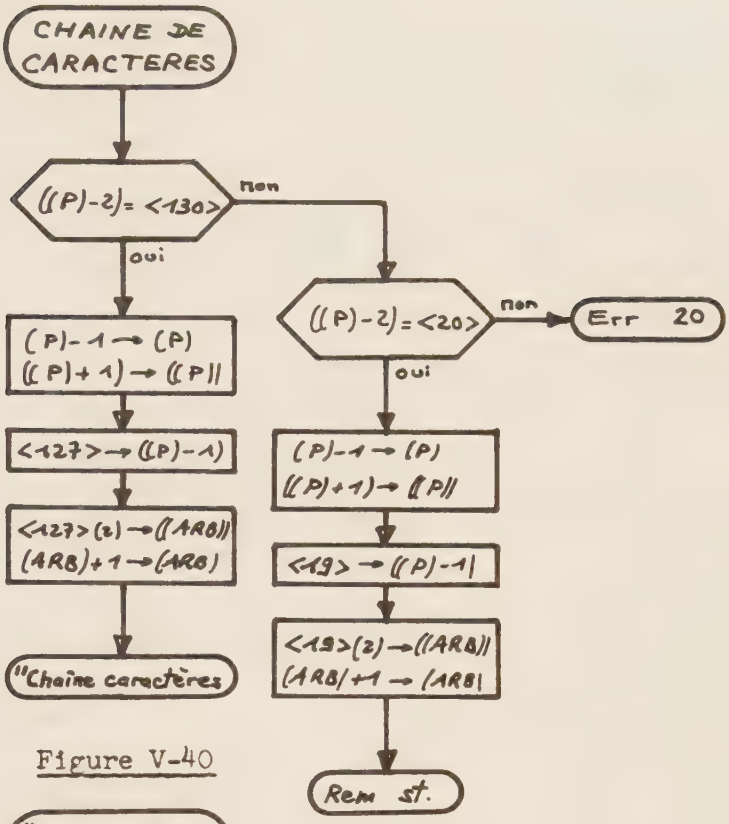


Figure V-40

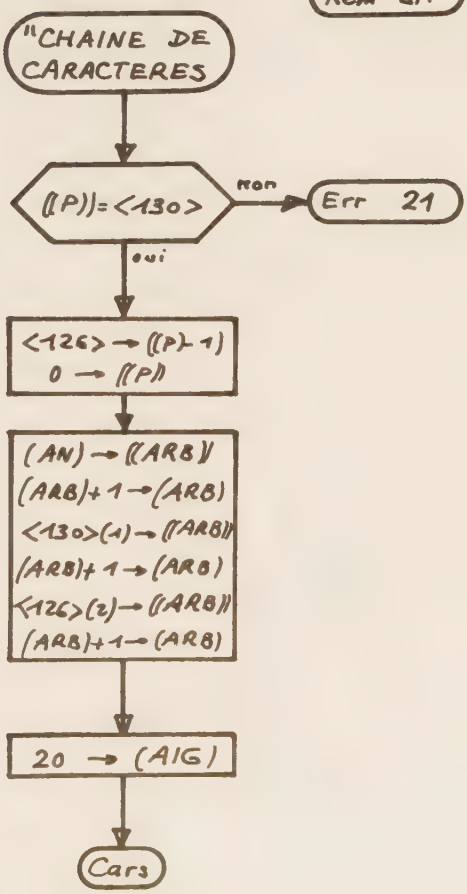


Figure V-41

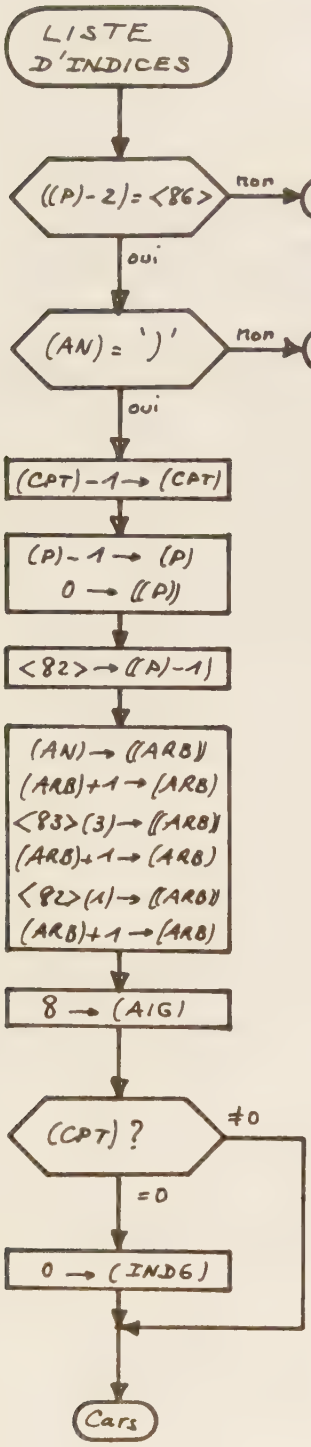


Figure V-42

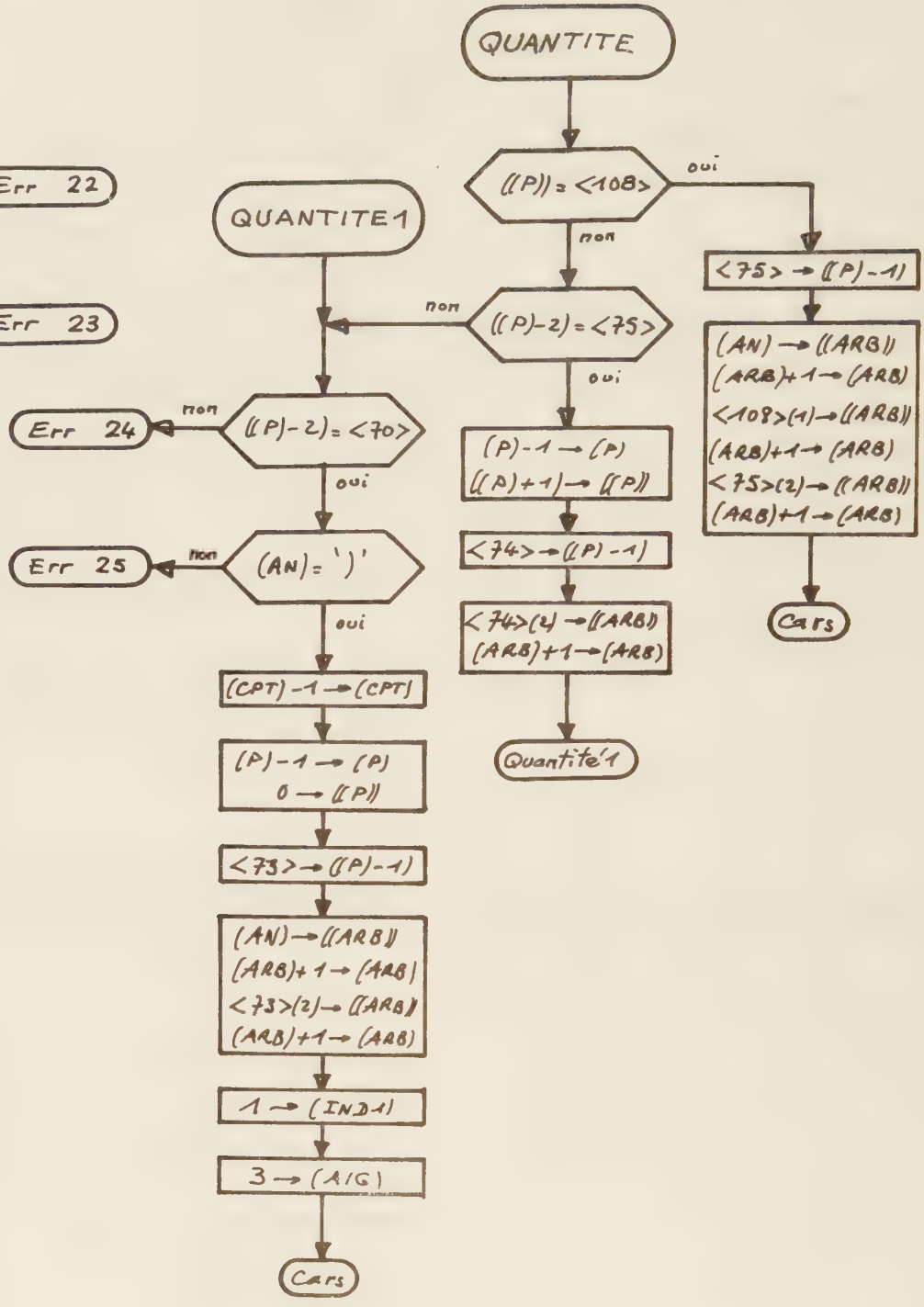


Figure V-43

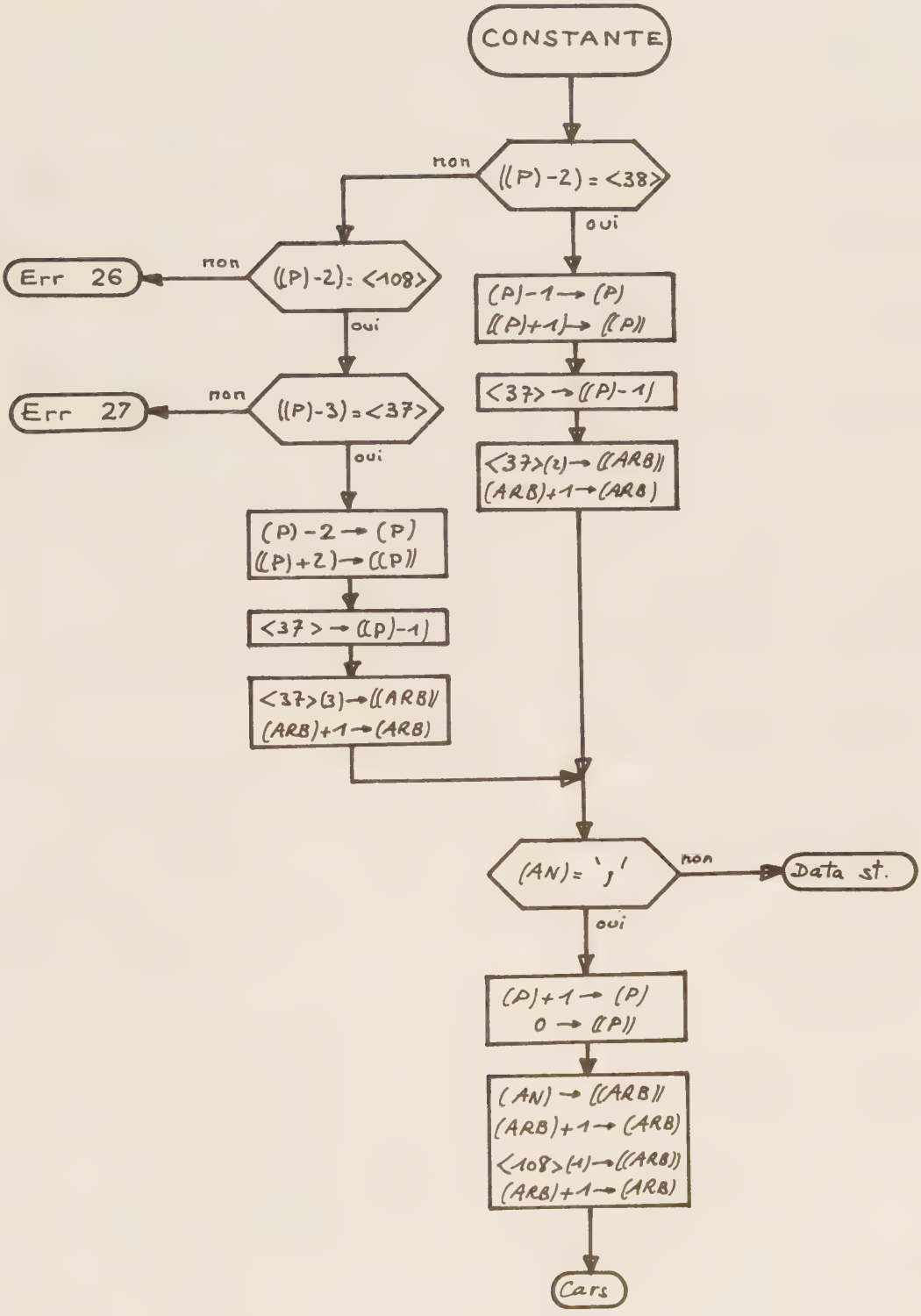

CONSTANTE
non
((P)-2) = <38>
oui
non
Err 26
((P)-2) = <108>
oui
(P)-1 → (P)
((P)+1) → ((P))
<37> → ((P)-1)
<37>(2) → ((ARB))
(ARB)+1 → (ARB)
non
Err 27
((P)-3) = <37>
oui
(P)-2 → (P)
((P)+2) → ((P))
<37> → ((P)-1)
<37>(3) → ((ARB))
(ARB)+1 → (ARB)
(AN) = 'j'
non
Data st.
oui
(P)+1 → (P)
0 → ((P))
(AN) → ((ARB))
(ARB)+1 → (ARB)
<108>(1) → ((ARB))
(ARB)+1 → (ARB)
Cars

Figure V-44

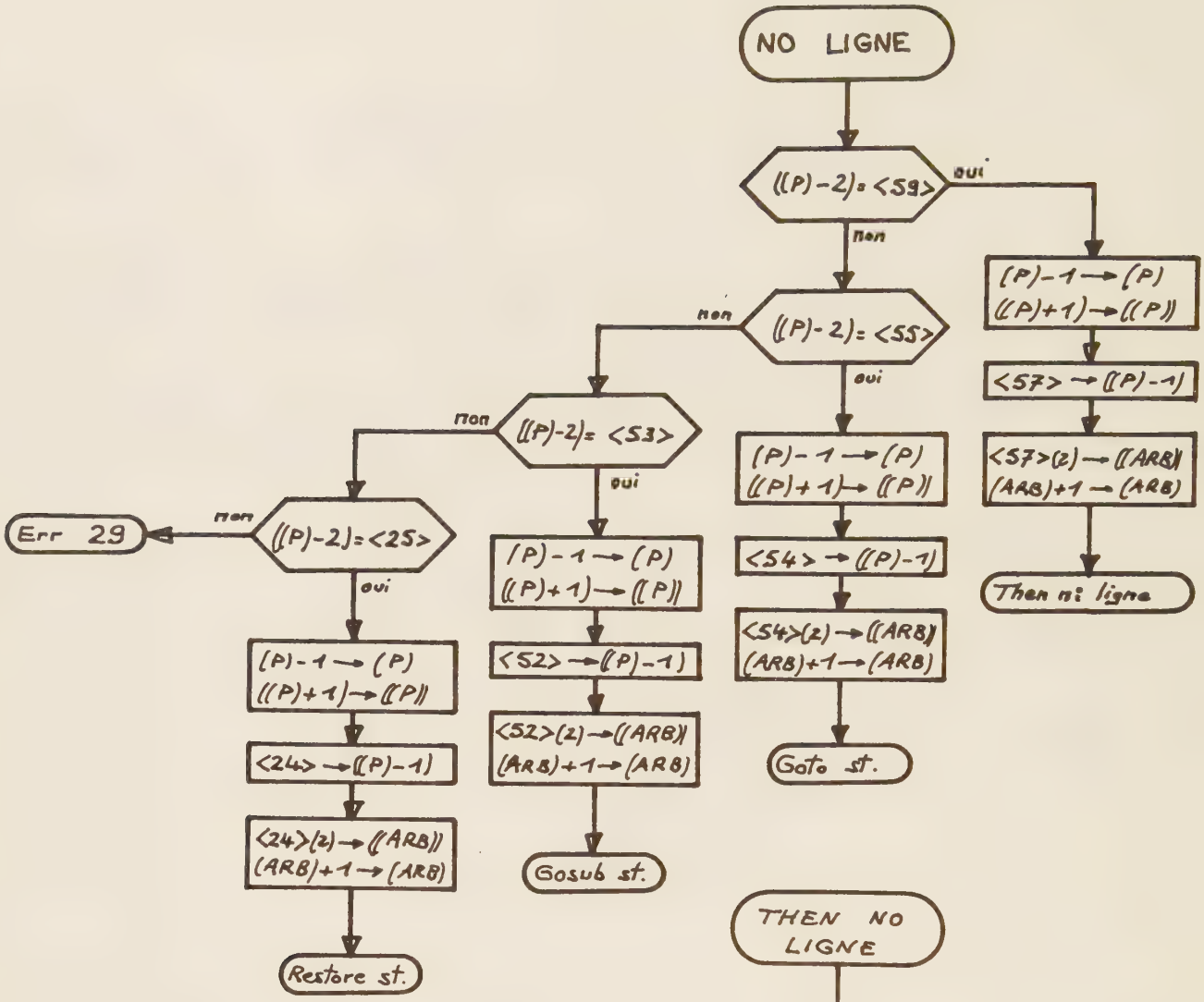


Figure V-45

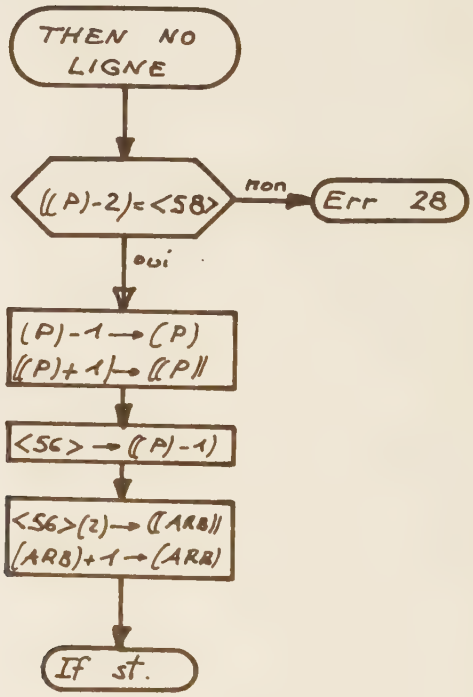


Figure V-46

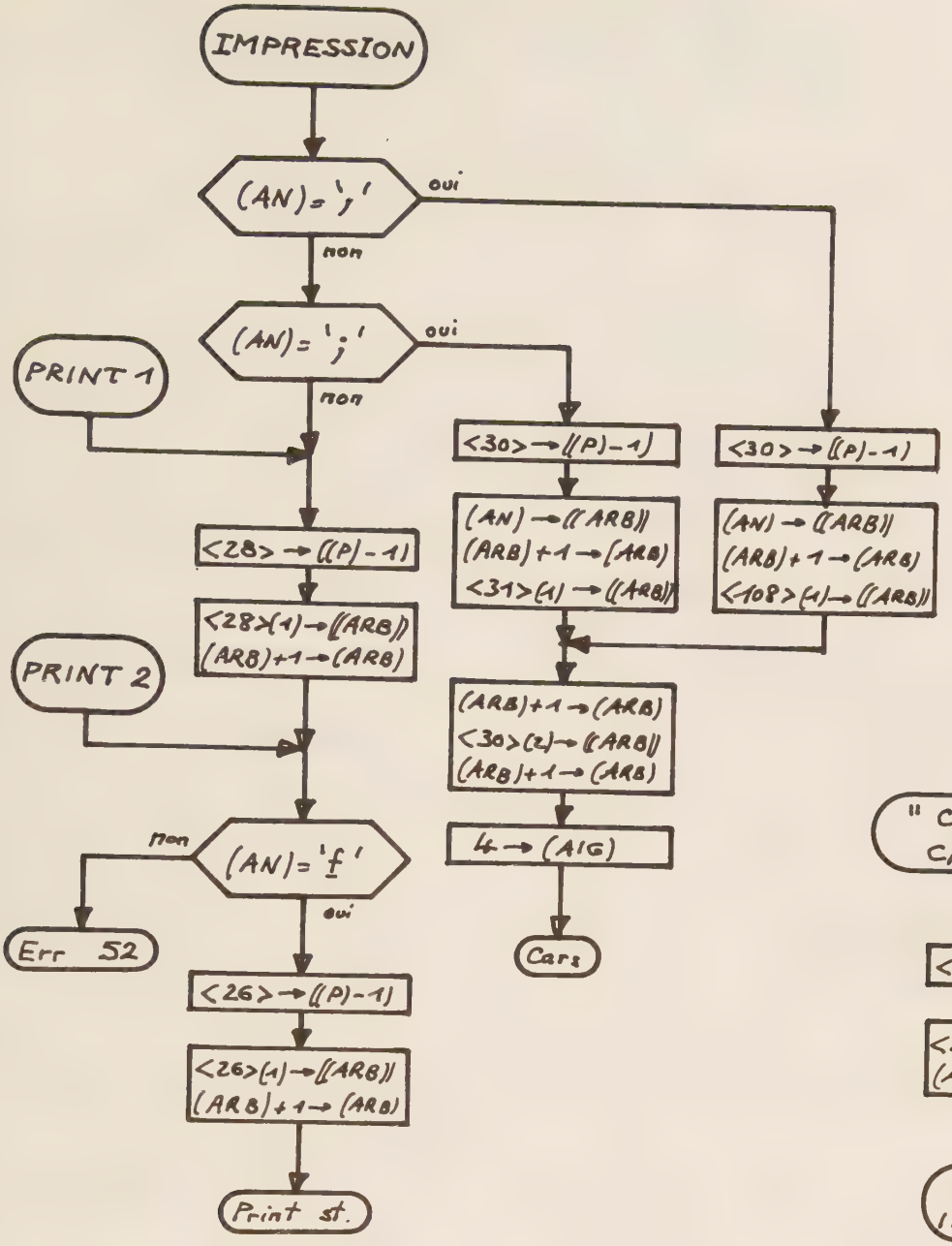


Figure V-48

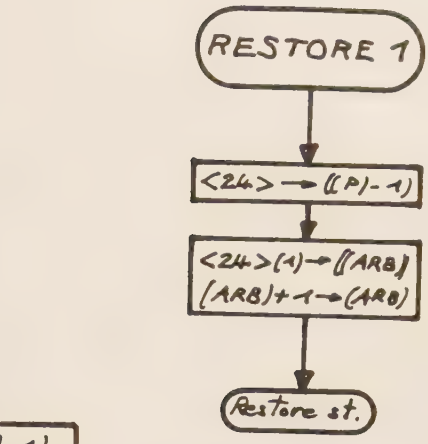


Figure V-47

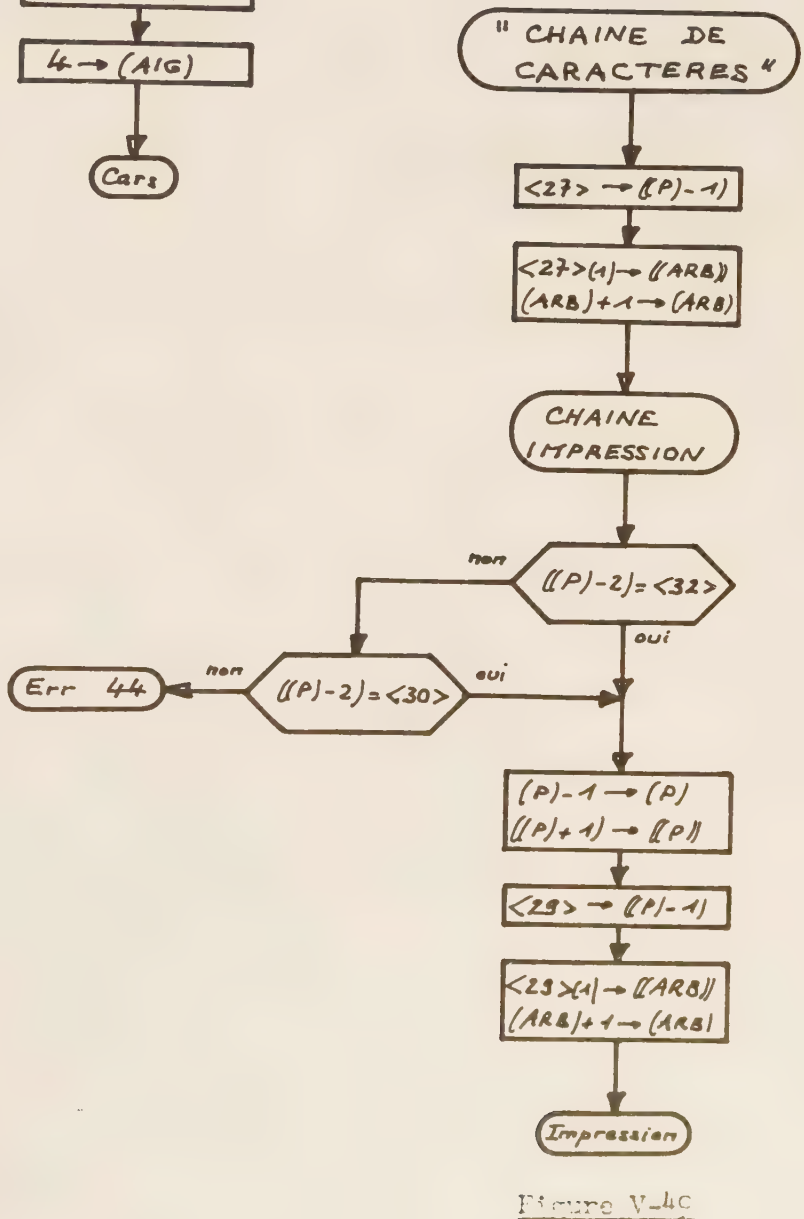
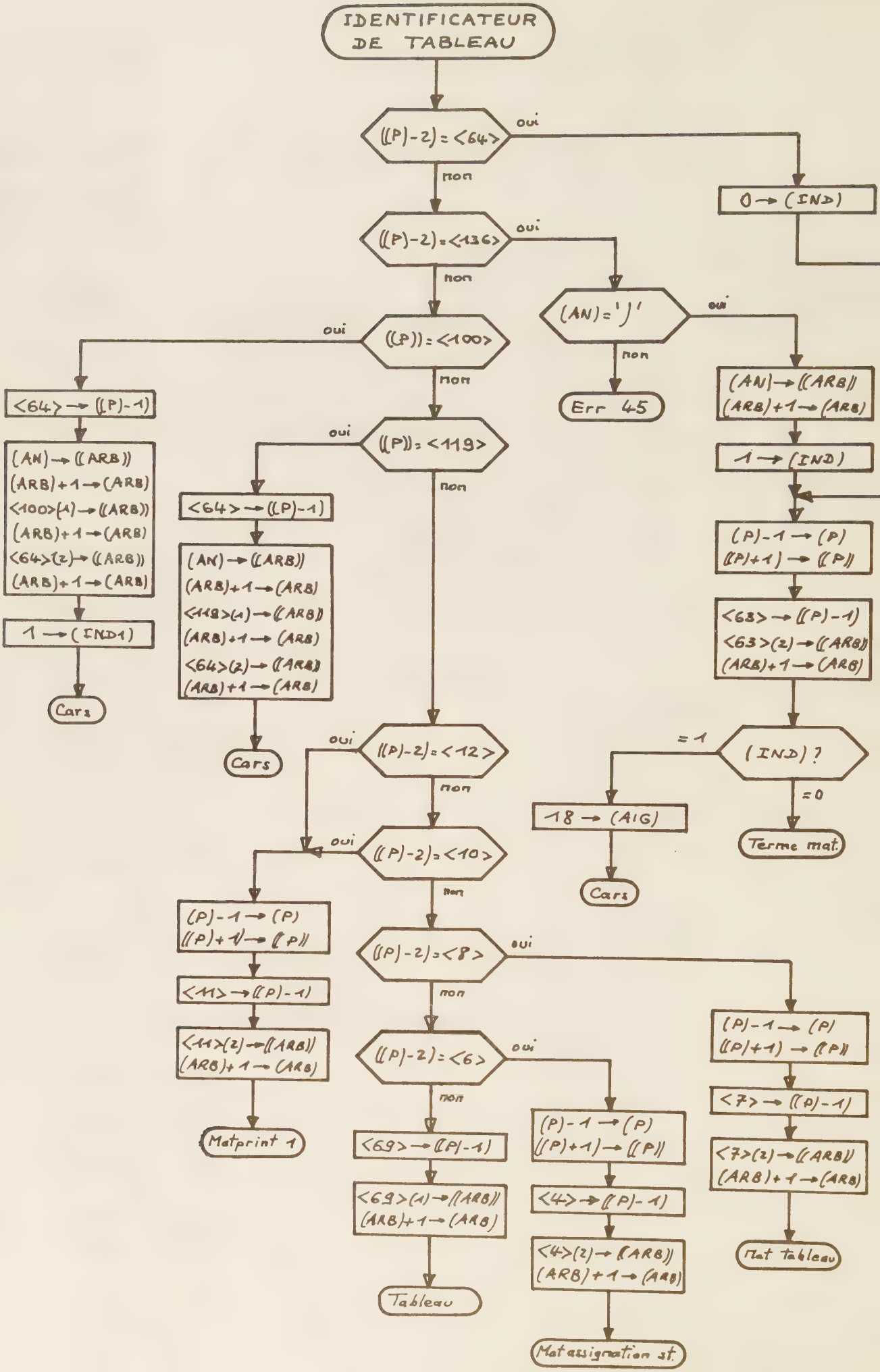
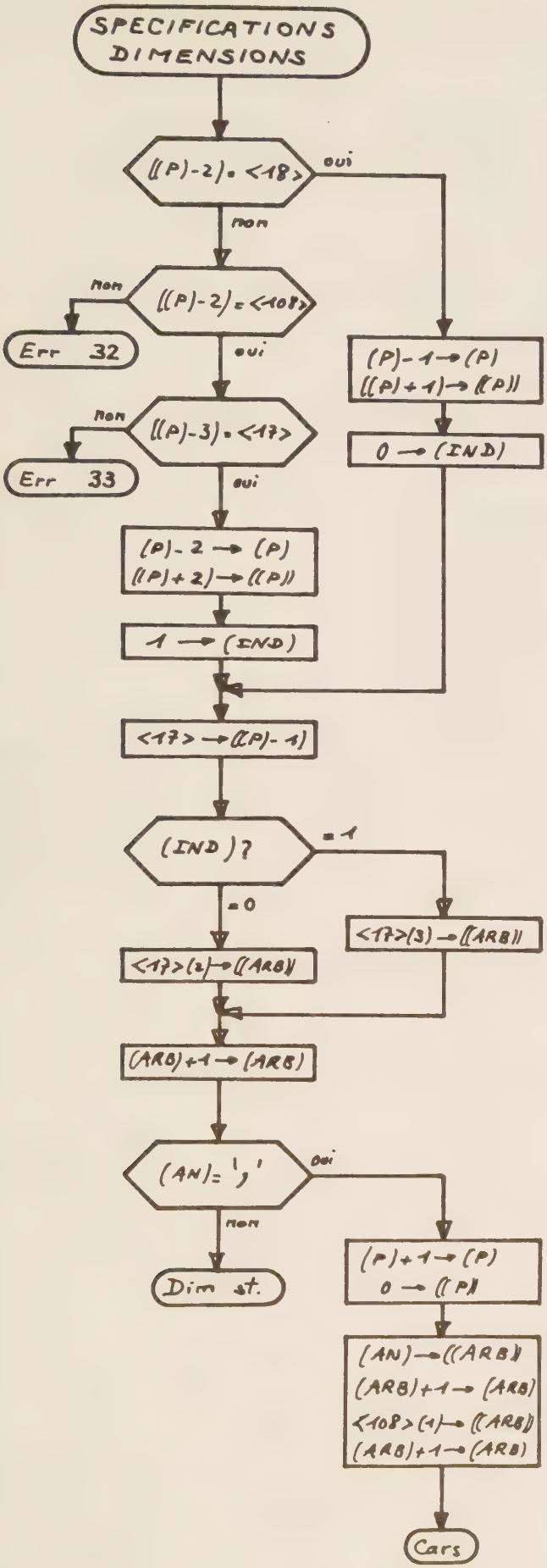


Figure V-49


IDENTIFICATEUR DE TABLEAU
((P)-2) = <64>
oui
non
0 → (IND)
((P)-2) = <136>
oui
non
(AN) = ')'
oui
non
Err 45
((P)) = <100>
oui
non
<64> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<100>(1) → ((ARB))
(ARB)+1 → (ARB)
<64>(2) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND1)
Cars
((P)) = <119>
oui
non
<64> → ((P)-1)
(AN) → ((ARB))
(ARB)+1 → (ARB)
<119>(1) → ((ARB))
(ARB)+1 → (ARB)
<64>(2) → ((ARB))
(ARB)+1 → (ARB)
Cars
(AN) → ((ARB))
(ARB)+1 → (ARB)
1 → (IND)
(P)-1 → (P)
((P)+1) → ((P))
<63> → ((P)-1)
<63>(2) → ((ARB))
(ARB)+1 → (ARB)
(IND) ?
=1
=0
18 → (AIG)
Cars
Terme mat
((P)-2) = <12>
oui
non
((P)-2) = <10>
oui
non
(P)-1 → (P)
((P)+1) → ((P))
<11> → ((P)-1)
<11>(2) → ((ARB))
(ARB)+1 → (ARB)
Matprint 1
((P)-2) = <8>
oui
non
(P)-1 → (P)
((P)+1) → ((P))
<7> → ((P)-1)
<7>(2) → ((ARB))
(ARB)+1 → (ARB)
Mat tableau
((P)-2) = <6>
oui
non
<69> → ((P)-1)
<69>(1) → ((ARB))
(ARB)+1 → (ARB)
Tableau
(P)-1 → (P)
((P)+1) → ((P))
<4> → ((P)-1)
<4>(2) → ((ARB))
(ARB)+1 → (ARB)
Mat assignation st.



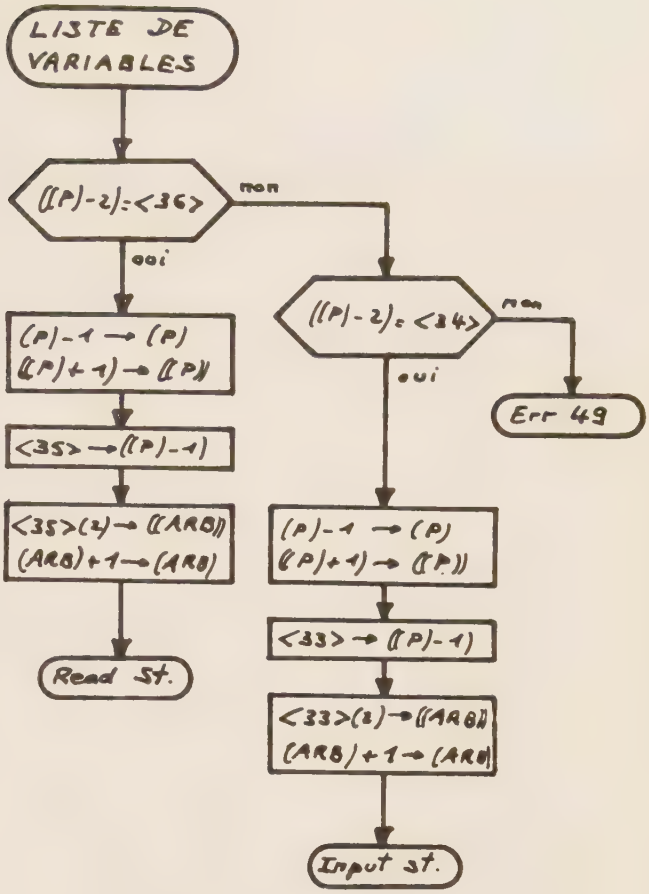


Figure V-51

Figure V-52

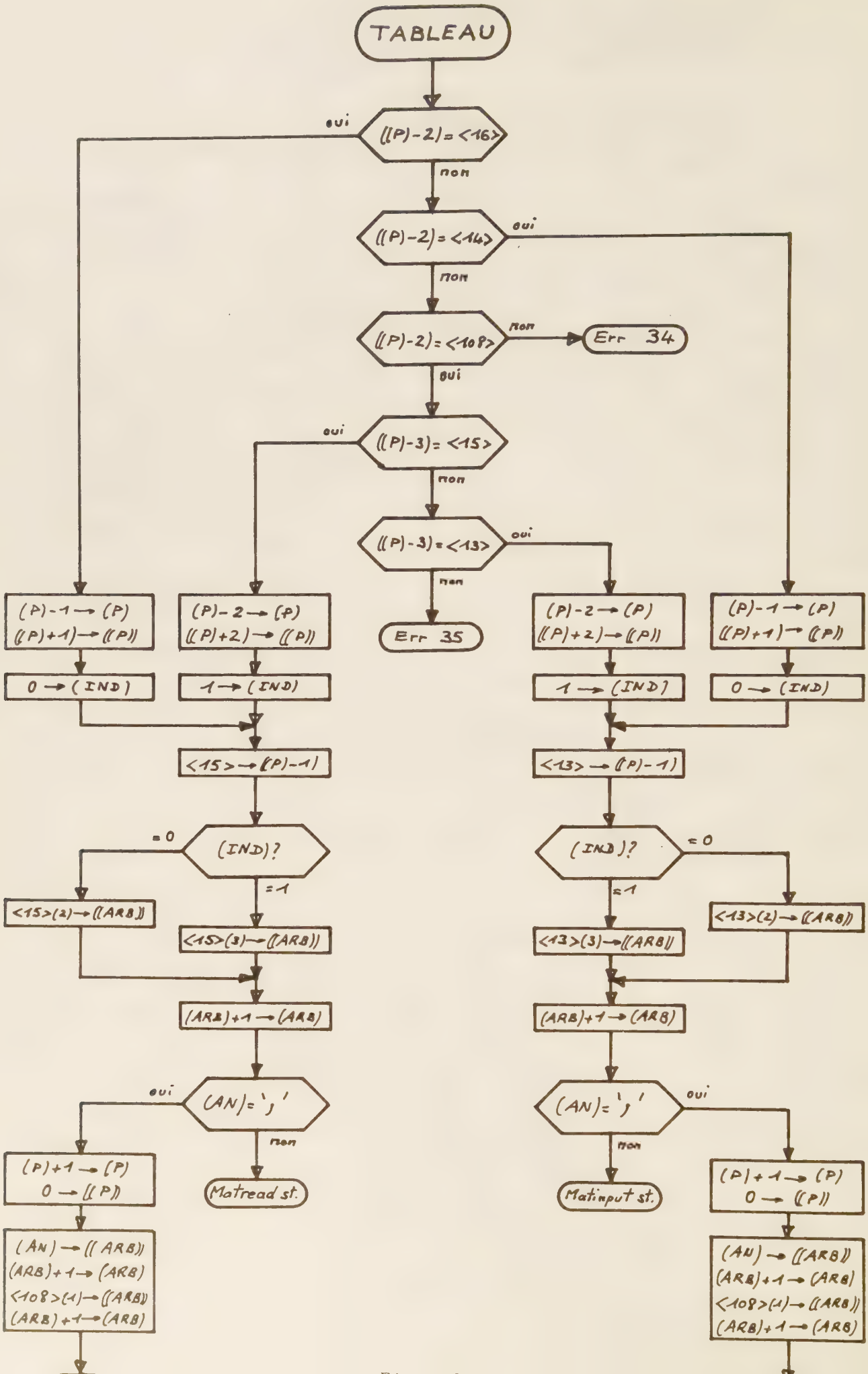


TABLEAU
((P)-2)=<16>
oui
non
((P)-2)=<14>
oui
non
((P)-2)=<10?>
non
Err 34
oui
((P)-3)=<15>
oui
non
((P)-3)=<13>
oui
non
Err 35
(P)-1→(P)
((P)+1)→((P))
0→(IND)
(P)-2→(P)
((P)+2)→((P))
1→(IND)
(P)-2→(P)
((P)+2)→((P))
1→(IND)
(P)-1→(P)
((P)+1)→((P))
0→(IND)
<15>→((P)-1)
<13>→((P)-1)
(IND)?
=0
=1
(IND)?
=0
=1
<15>(2)→((ARB))
<15>(3)→((ARB))
<13>(2)→((ARB))
<13>(3)→((ARB))
(ARB)+1→(ARB)
(ARB)+1→(ARB)
(AN)='j'
oui
non
(AN)='j'
oui
non
(P)+1→(P)
0→((P))
Matread st.
Matinput st.
(P)+1→(P)
0→((P))
(AN)→((ARB))
(ARB)+1→(ARB)
<108>(1)→((ARB))
(ARB)+1→(ARB)
(AN)→((ARB))
(ARB)+1→(ARB)
<108>(1)→((ARB))
(ARB)+1→(ARB)

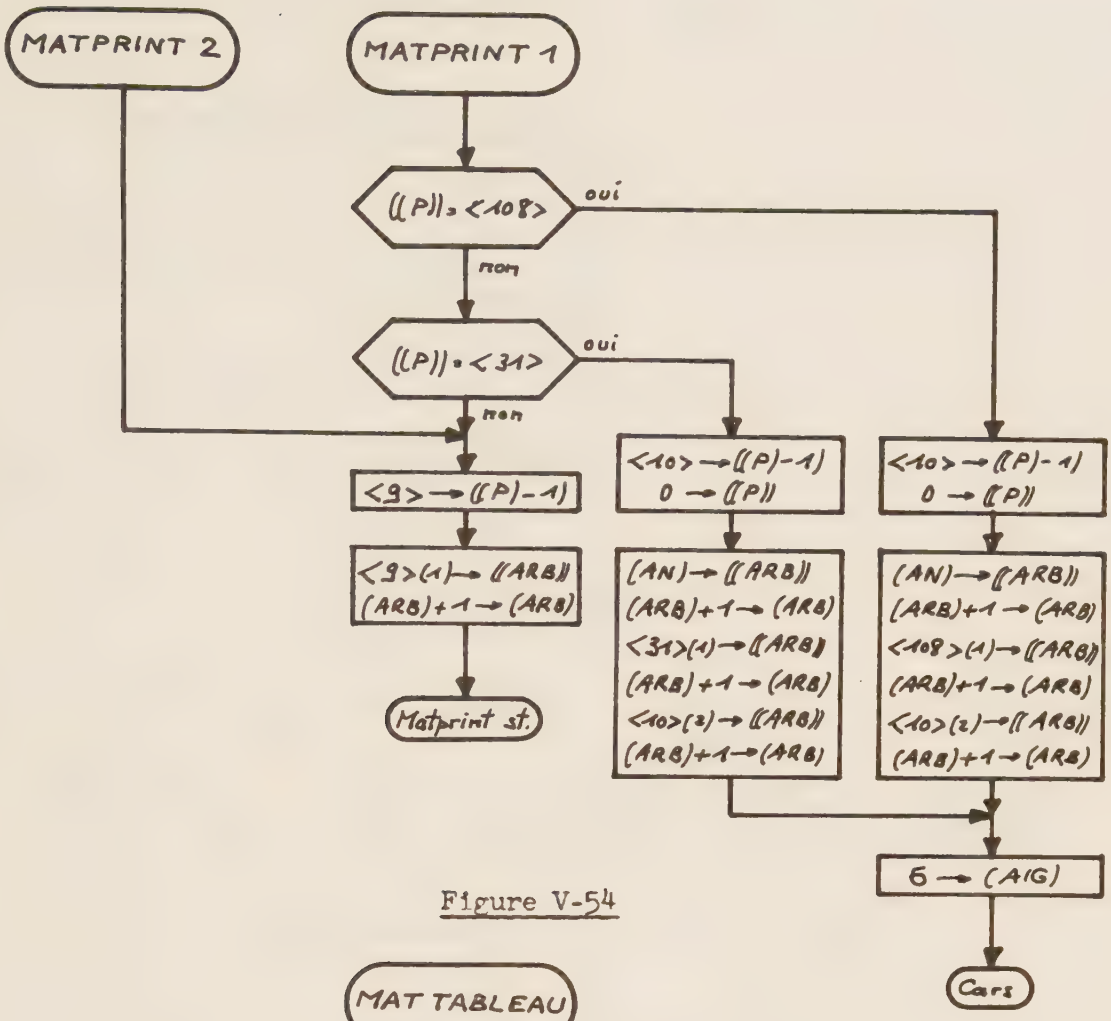


Figure V-54

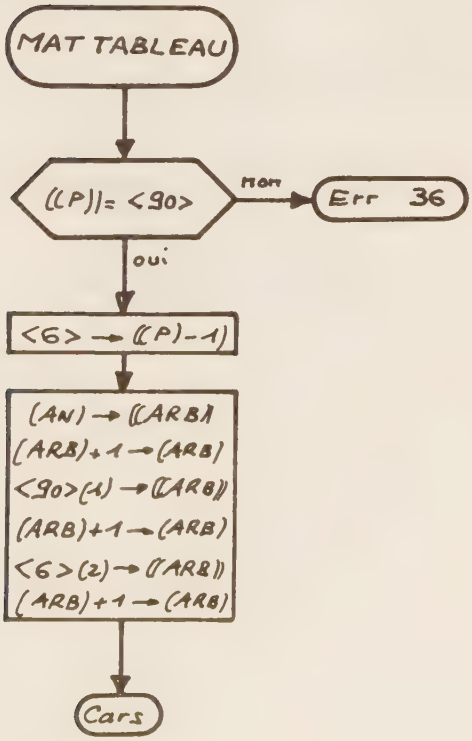


Figure V-55

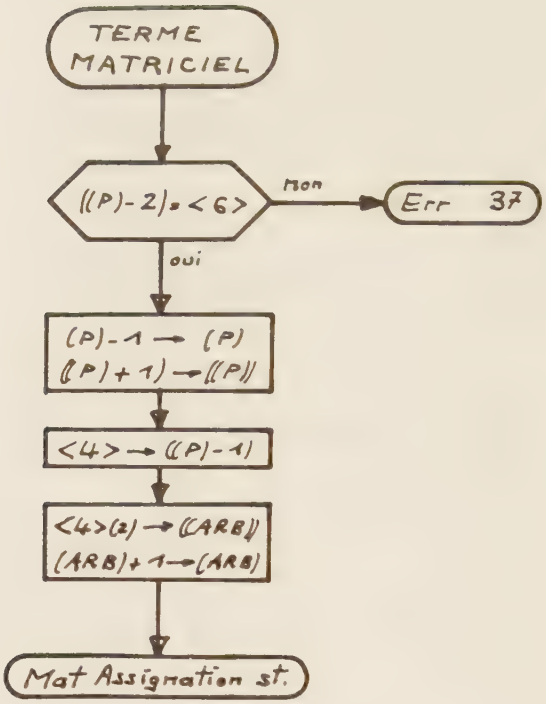


Figure V-56

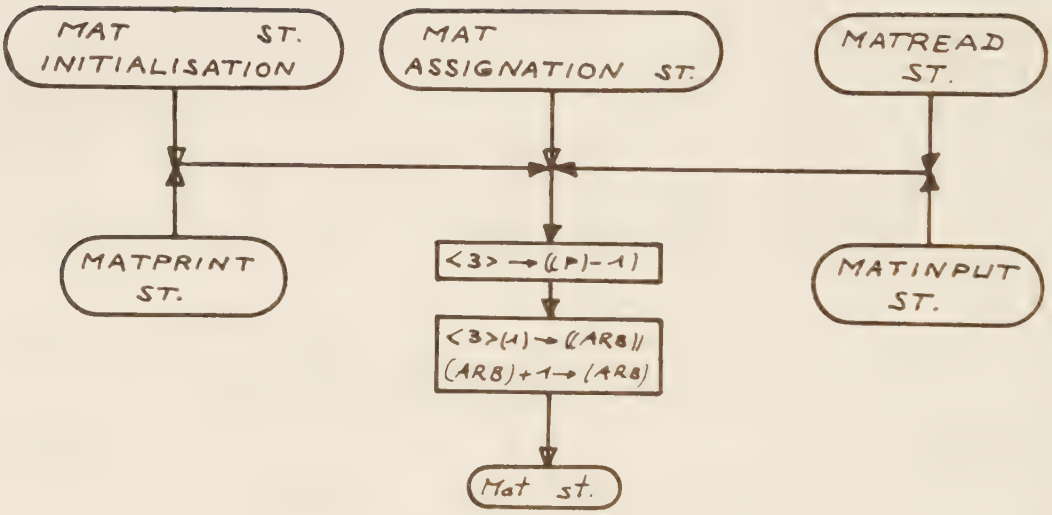


Figure V-57

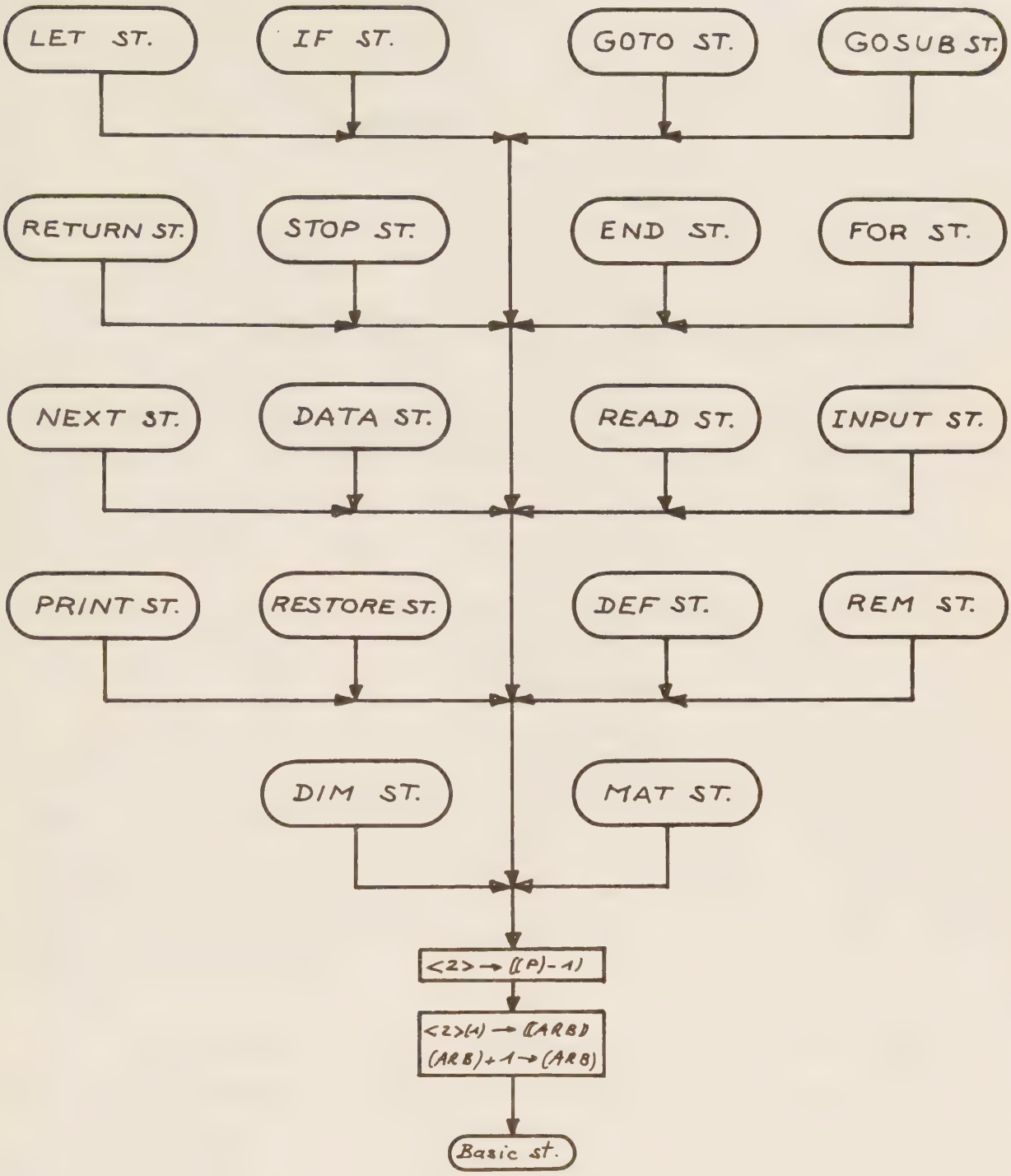

LET ST.
IF ST.
GOTO ST.
GOSUB ST.
RETURN ST.
STOP ST.
END ST.
FOR ST.
NEXT ST.
DATA ST.
READ ST.
INPUT ST.
PRINT ST.
RESTORE ST.
DEF ST.
REM ST.
DIM ST.
MAT ST.
<2>→((P)-1)
<2>(4)→((ARB))
(ARB)+1→(ARB)
Basic st.

Figure V-58

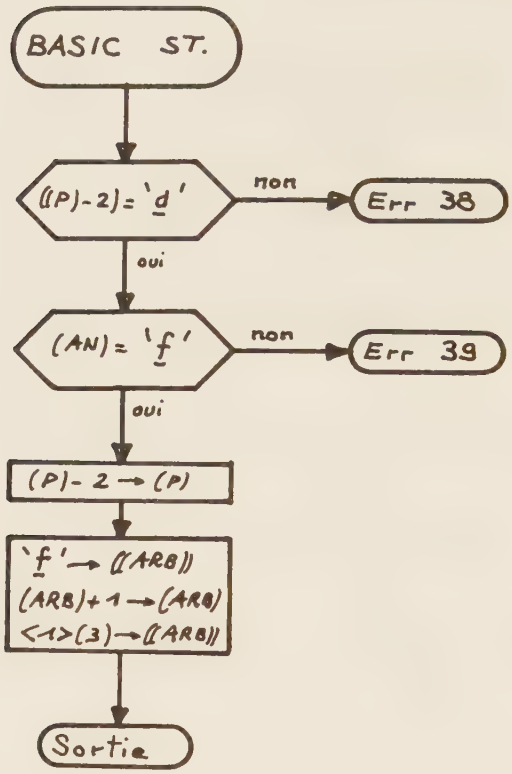

BASIC ST.
((P)-2)='d'
non
Err 38
oui
(AN)='f'
non
Err 39
oui
(P)-2→(P)
'f'→((ARB))
(ARB)+1→(ARB)
<1>(3)→((ARB))
Sortie

Figure V-59

CHAPITRE VI

INTERPRETEUR

VI-1 - ANALYSE PROGRAMME

L'analyseur que nous venons de décrire permet de faire l'analyse syntaxique des instructions prises une à une. Si nous considérons comme relevant de la syntaxe, le fait que des boucles doivent être correctement emboîtées, que le dernier ordre du programme doit être un ordre END, alors il nous faut, avant de passer à la phase exécution, terminer l'analyse globale du programme.

L'organigramme représenté à la figure VI-1, nous montre de quelle façon cette analyse est effectuée.

Le programme BASIC est supposé disponible sous la forme d'un fichier de cartes obtenues par un ordre de perforation "PUNCH" quand toutes les instructions sont entrées et ont été analysées sans erreur. Un carte fin de fichier permet de savoir que le programme est entré complètement. Il est important de vérifier que les cartes du programme sont correctement classées, sinon il y a affichage de :::::: sur l'unité de visualisation.

Les instructions qui nous préoccupent sont les instructions FØR (début de boucle) et NEXT (fin de boucle). A la reconnaissance d'un "FOR", on recherche quelle est la variable de boucle, qu'on introduit dans la pile "LØØP". A la reconnaissance d'un "NEXT", on cherche la variable citée, dont on compare le nom au dernier mis dans la pile "LØØP". Si la comparaison se révèle négative, on sort en erreur ; dans le cas contraire on descend la pile d'un cran.

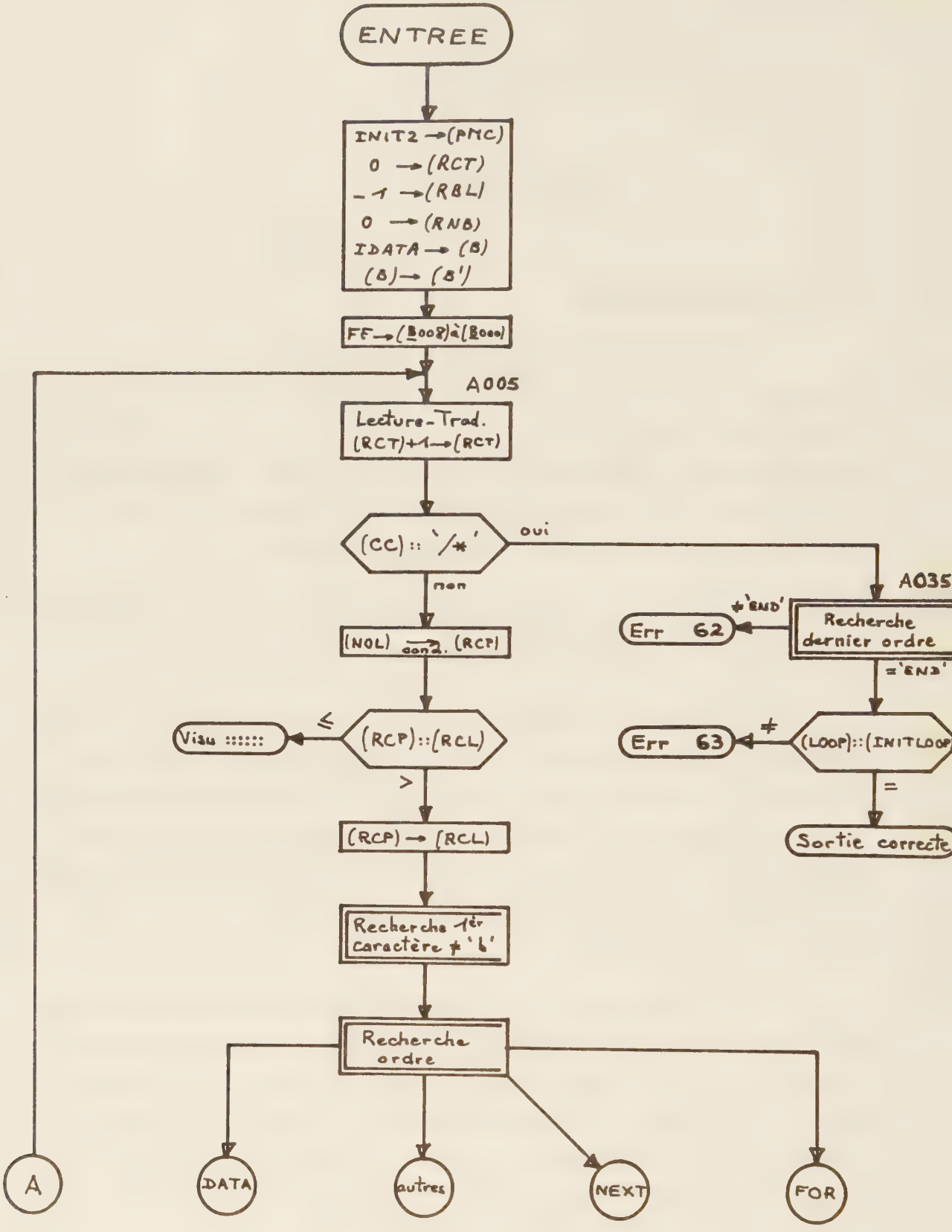

ENTREE
INIT2 → (PMC)
0 → (RCT)
-1 → (RBL)
0 → (RNB)
IDATA → (B)
(B) → (B')
FF → (B008) à (B000)
A005
Lecture-Trad.
(RCT)+1 → (RCT)
(CC) :: '/*'
oui
non
A035
Recherche dernier ordre
≠ 'END'
Err 62
= 'END'
(NOL) → (RCP)
(RCP) :: (RCL)
≤
Visu ::::::
≠
Err 63
(LOOP) :: (INITLOOP)
=
Sortie correcte
>
(RCP) → (RCL)
Recherche 1er caractère ≠ 'b'
Recherche ordre
A
DATA
autres
NEXT
FOR

Figure VI-1

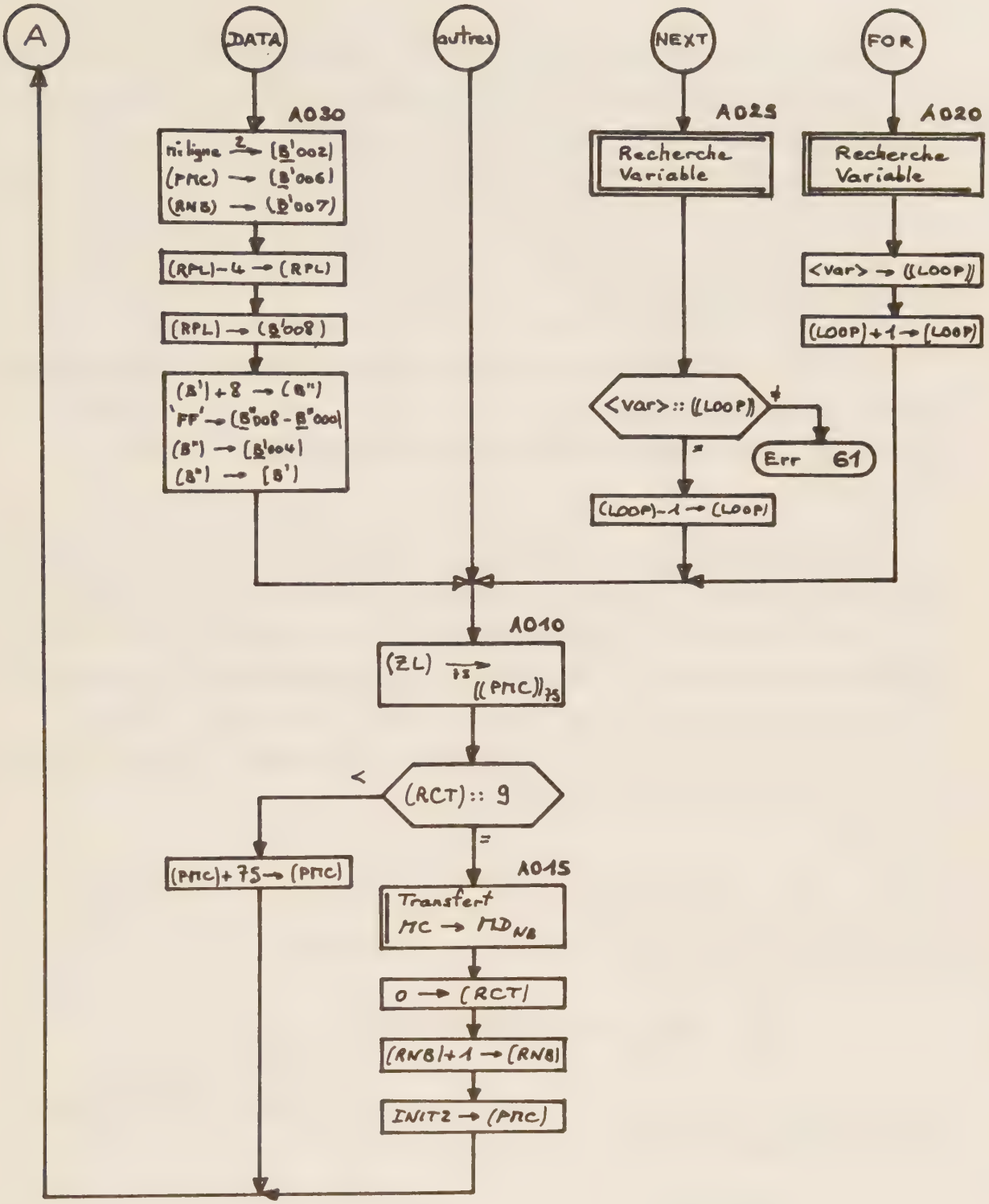

A
DATA
autres
NEXT
FOR
A030
A025
A020
ni ligne → (B'002)
(PTC) → (B'006)
(RNB) → (B'007)
(RPL)-4 → (RPL)
(RPL) → (B'008)
(B') + 8 → (B")
'FF' → (B"008 - B"000)
(B") → (B'004)
(B') → (B')
Recherche Variable
Recherche Variable
<var> → ((LOOP))
(LOOP)+1 → (LOOP)
<var> :: ((LOOP))
+
=
Err 61
(LOOP)-1 → (LOOP)
A010
(ZL) → ((PTC))75
<
(RCT) :: 9
=
(PTC)+75 → (PTC)
A015
Transfert MC → MD NB
0 → (RCT)
(RNB)+1 → (RNB)
INIT2 → (PTC)

Figure VI-2

A la détection de la carte fin de fichier, on vérifie que le dernier ordre est un "END", puis que la pile "LØØP" est revenue à l'état initial, sinon, il y a un ordre "FØR" sans ordre "NEXT" correspondant.

VI-2 - TRAITEMENT DES "DATA"

Les seules données entrant dans un programme BASIC sont des données numériques. Elles peuvent être disponibles sous la forme de constantes numériques séparées par des virgules et figurant dans un ordre DATA.

En remarquant qu'une donnée appartenant à un DATA n'est généralement pas utilisée de nombreuses fois, il semble inutile de faire une pré-compilation des éléments des DATA. Nous ne déclencherons la conversion "donnée DATA" ⟶ "constante en flottant", qu'à chaque ordre de lecture dans un "DATA" (ordre READ). Cette méthode permet d'économiser toute la place qu'il faudrait pour stocker l'ensemble des données transformées par pré-compilation.

Il s'agit néanmoins de savoir, lors d'une conversion, si l'on est au début d'une ligne DATA, au milieu, ou à la fin, ceci oblige à conserver en mémoire un certain nombre d'informations.

VI-2-1 - Mode d'enregistrement du programme BASIC

Il est clair, que la façon dont on va traiter les blocs DATA, va dépendre du mode d'enregistrement du programme.

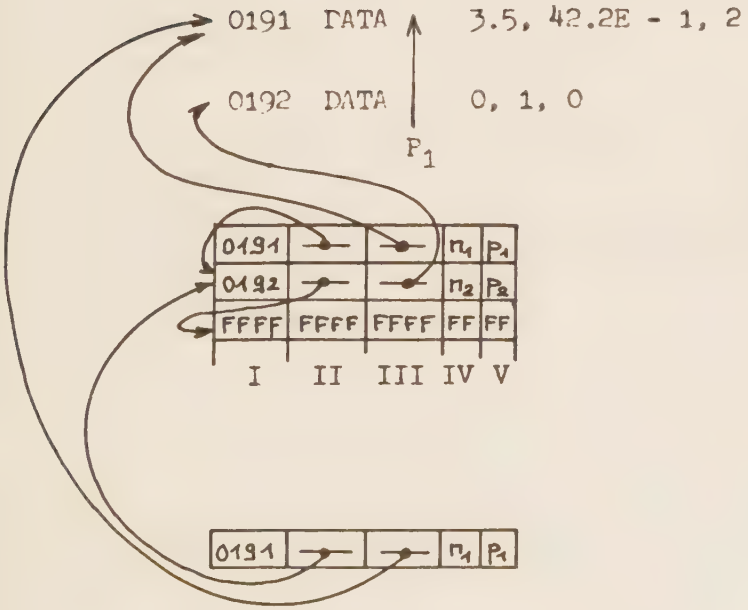
On sait qu'une ligne de télétype peut faire 72 caractères. Nous avons ici, augmenté à 75 le nombre de caractères, pour des raisons de commodité pratique relatives au maniement des numéros de ligne.

Etant hors de question, pour des raisons d'encombrement, de conserver tout le programme BASIC en mémoire (c'est également vrai en analyse, voir annexe B), il faut utiliser le tambour, on peut alors mettre jusqu'à 9 instructions BASIC par piste ($9 \times 75 = 675 < 700$ octets), les instructions sont enregistrées en séquence dans leur ordre d'arrivée (qui est l'ordre des numéros de ligne croissants). A chaque bloc de 9 instructions, on commande l'enregistrement sur une piste du tambour, jusqu'à épuisement des instructions. On conserve le numéro de la dernière piste programme dans un registre réservé à cet effet (registre RNB).

VI-2-2 - Organisation du traitement des DATA

Les lignes "DATA" peuvent être écrites n'importe où dans le programme, il est nécessaire, pour éviter une perte de temps importante, de repérer dans le programme les numéros de toutes les lignes comportant un "DATA". On constitue un bloc de renseignements de type enchaîné, contenant tous les renseignements nécessaires à l'extraction des données.

Prenons un exemple :



I = n° de ligne du "DATA"

II = pointeur d'enchaînement dans le bloc

III - adresse en MC de la ligne "DATA"

IV - n_i, n° de la piste qui contient le "DATA" du programme enregistré

V - P_i, pointeur relatif au début de ligne, contenant l'adresse de la première position qui suit le mot-clé DATA.

Cellule de recherche

Figure VI-3

On obtient l'adresse nécessaire au programme de conversion en additionnant (III) et (V).

Le bloc de renseignements pouvant avoir une taille importante 8 x (N+1) octets, on préfère le conserver en tambour, et avoir en permanence en mémoire centrale une cellule de recherche, qui, tour à tour se trouve être une des cellules du bloc.

Tous les renseignements inclus dans cette cellule, sont relatifs au "DATA" courant.

Remarquons qu'il faut enchaîner dans le bloc, une ligne spéciale marquée FF ... FF, dont la détection en exécution, indique que l'on va chercher des données alors que les "DATA" sont épuisés, ce qui provoque une sortie en erreur (OUT OF DATA).

A l'instant d'une lecture dans un "DATA", on récupère le n° de ligne du "DATA" courant, on va lire la piste dont le numéro est indiqué dans la cellule courante, on prélève la ligne "DATA" que l'on stocke dans une zone réservée à la conversion. On se trouve, par exemple dans l'état suivant :

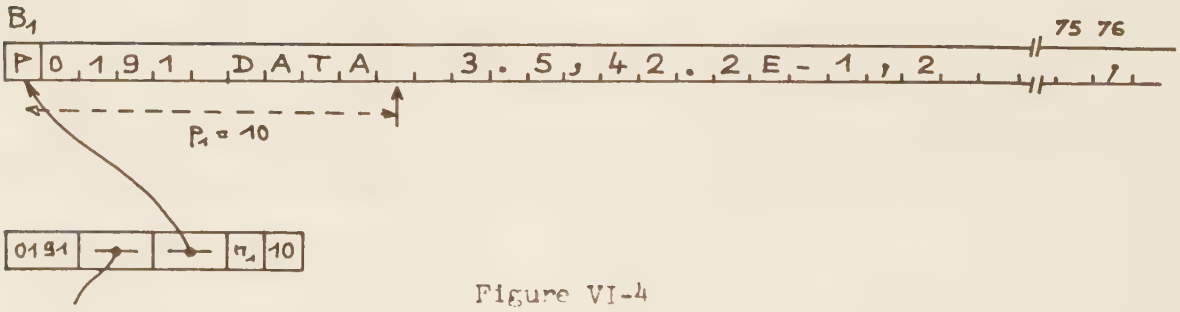


Figure VI-4

A l'appel du sous-programme de conversion, on fournit l'adresse de départ de conversion, en additionnant P_1 à l'adresse de base B_1. La conversion effectuée, on dispose de la constante en flottant dans un registre accumulateur, et l'adresse de la position qui suit la première virgule rencontrée (adresse du 4), dans un autre registre.

On peut comparer cette adresse, avec l'adresse de la virgule repérée 76 relativement à B_1, si on trouve davantage, c'est que la ligne "DATA" est épuisée, et il faut remettre à jour la cellule de recherche en déroulant un coup le chaînage, sinon, il suffit de remettre à jour P_i dans la cellule de recherche.

Un ordre "RESTØRE", sans précision d'un numéro de ligne signifie qu'on reprend les DATA au début, il suffit de recopier la première cellule du bloc de renseignements dans la cellule de recherche.

Si un numéro de ligne est précisé, il faut recopier la cellule du bloc comportant le même numéro.

La création du bloc de renseignements est effectuée en même temps que l'analyse-programme, au fur et à mesure de la lecture du programme BASIC.

L'utilisation de la cellule de recherche et du bloc de renseignements sera vue sur les organigrammes relatifs aux ordres READ et RESTØRE.

VI-3 - TRAITEMENT DES VARIABLES SIMPLES

VI-3-1 - Organisation en MC

Une variable simple en BASIC peut se présenter de 2 façons, soit une lettre seule, soit une lettre suivie d'un chiffre. Il y a donc

286 variables simples différentes au maximum. La représentation en flottant interne requérant 5 octets par variable, on peut penser réserver une table de 5 × 286 = 1430 octets, dans laquelle on accède facilement par le caractère alphabétique et par le caractère numérique (s'il existe). La longueur de la table est trop importante relativement à la taille MC disponible pour que nous retenions cette solution.

Du point de vue encombrement, la solution la plus économique, consiste à empiler les variables au fur et à mesure de leur apparition dans les instructions BASIC, le problème est alors de les rechercher là où il faut. Si l'empilement est fait séquentiellement, sans aucun classement préalable, la recherche est également séquentielle, le temps d'accès moyen à une variable étant alors le temps qu'il faut pour accéder à la variable du milieu de la pile.

Encombrement de la pile : 7 × N octets, si N est le nombre de variables. Il faut 2 octets pour conserver le nom de 2 caractères et 5 octets pour la valeur numérique affectée. A partir de 204 variables (1430/7 = 204), l'encombrement devient supérieur à la table évoquée ci-dessus, il semble néanmoins raisonnable de penser qu'on n'atteindra pas un nombre aussi grand de variables. Le temps d'accès est aléatoire, le temps d'accès moyen est bien supérieur à ce qu'on avait dans le premier cas.

Nous nous orientons alors vers un moyen terme. Réservons une table de longueur fixe égale à 572 octets soit 286 × 2, nous pouvons mettre dans chaque cellule de 2 octets représentant la variable, l'adresse d'une cellule de 5 octets qui contient la valeur numérique affectée à la variable.

L'encombrement est égal à 572 + 5×N, où N est le nombre de variables employées, le temps d'accès est rapide, car l'accès à une cellule de 2 octets de la table des variables est simple et calculable.

On remplit la table des variables de drapeaux "FF" au départ, qu'on remplace lors de la définition d'une variable par l'adresse de la cellule numérique correspondante, et qui permet en exécution de savoir si la variable citée a déjà été définie ou non.

L'ensemble des cellules numériques constitue la pile des valeurs numériques des variables simples dont le pointeur est PTV.

Exemple :

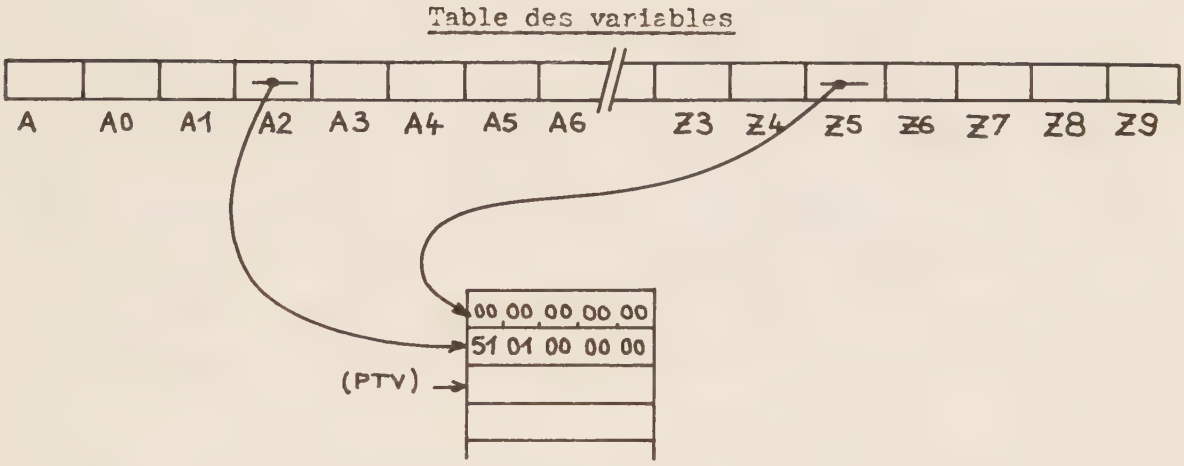
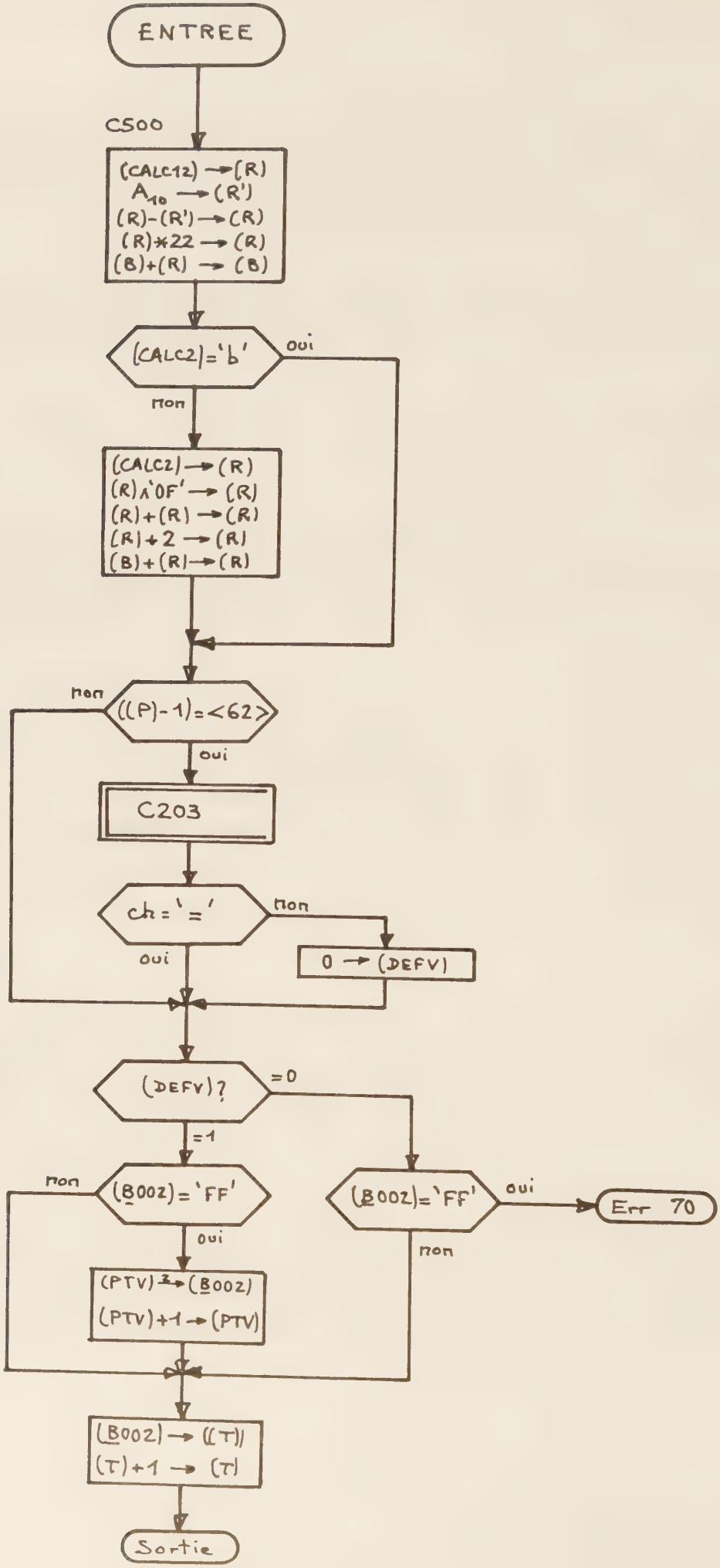


Figure VI-5

VI-3-2 - Sous-programme de traitement des variables simples

Si CALC12 et CALC2 contiennent respectivement la lettre et le chiffre qui permettent d'écrire la variable, DEFV étant un indicateur de définition de variable, B contenant l'adresse de tête de la table des variables, PTV étant le pointeur dans la pile des valeurs numériques, T le pointeur dans la pile interprétation (dont nous reparlerons), alors on peut utiliser l'organigramme de la figure VI-6, pour traiter les variables simples.


ENTREE
C500
(CALC12) → (R)
A10 → (R')
(R)-(R') → (R)
(R)*22 → (R)
(B)+(R) → (B)
(CALC2)='b'
oui
non
(CALC2) → (R)
(R)∧'0F' → (R)
(R)+(R) → (R)
(R)+2 → (R)
(B)+(R) → (R)
non
((P)-1)=<62>
oui
C203
ch='='
non
oui
0 → (DEFV)
(DEFV)?
=0
=1
non
(B002)='FF'
oui
(PTV) → (B002)
(PTV)+1 → (PTV)
(B002)='FF'
oui
Err 70
non
(B002) → ((T))
(T)+1 → (T)
Sortie

Nous verrons le rôle de la séquence qui commence par le test de ((P) - 1) à <62> dans la description de l'ordre LET.

DEFV est mis à 1, par les routines de traitement des mots-clés READ, INPUT, LET et FØR.

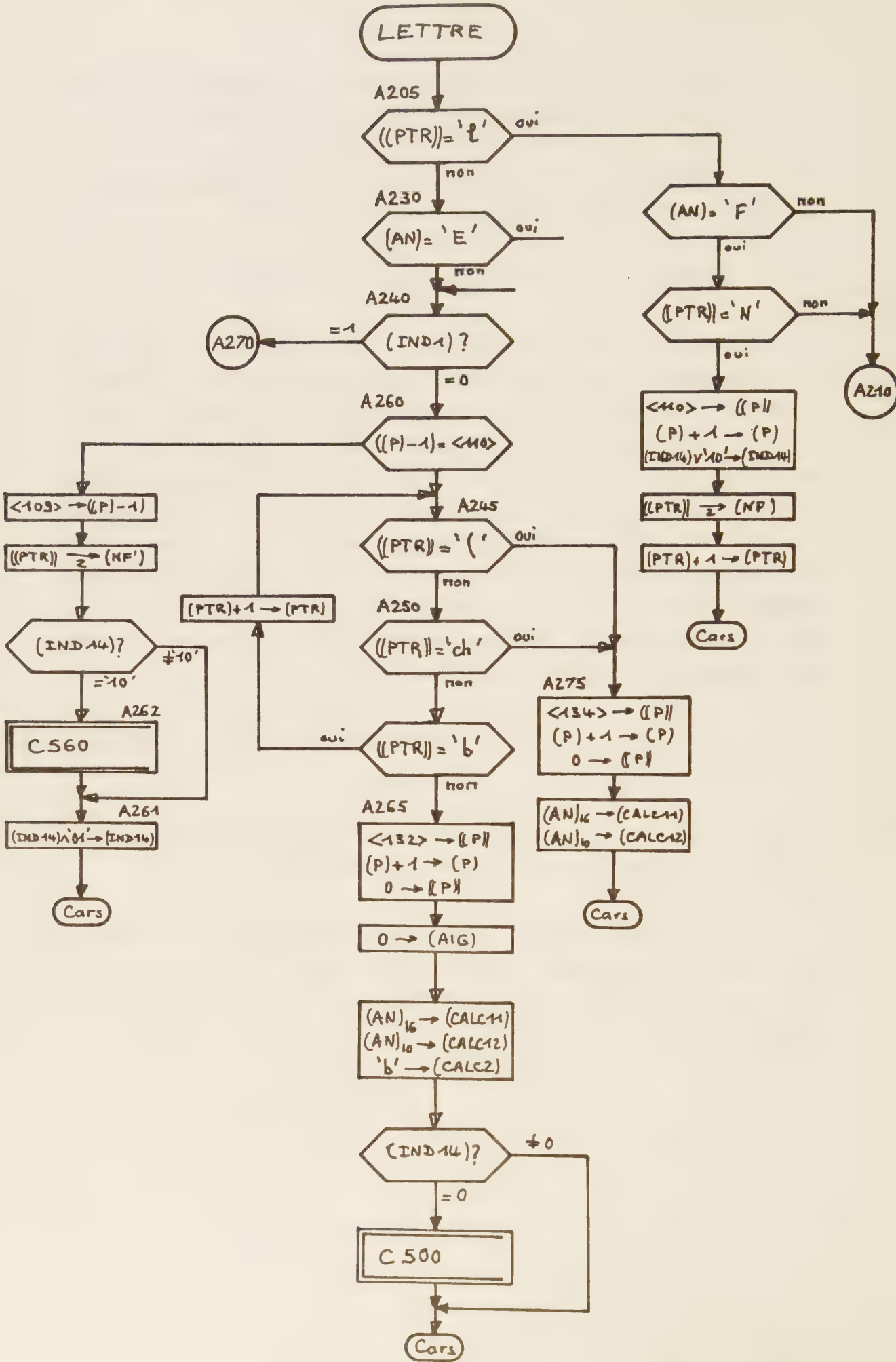
VI-3-3 - CALC12 et CALC2

On vient de voir que CALC1, CALC12 et CALC2 contiennent les 2 caractères constituant la variable simple, CALC2 contenant éventuellement un blanc. Il faut se poser le problème du chargement de ces 2 positions de mémoire. On sait que dans LETTRE (figure VI-7), on reconnaît la structure d'une variable simple en ne détectant pas un contexte matriciel, ni le radical FN des fonctions, ni la parenthèse d'ouverture en contexte postérieur, ni un chiffre. La routine utilisée est repérée par A265, on y charge CALC11 et CALC12 par les codes hexadécimal et décimal de la lettre considérée, on met CALC2 à blanc, et si le contexte fonction n'est pas détecté, on appelle le sous-programme de traitement des variables simples.

En A275, qui montre la reconnaissance d'une lettre seule, on charge seulement CALC11 et CALC12.

Remarque : dans certains cas, on a besoin du code hexadécimal, c'est la raison de la mise en garde de ce code dans CALC11.

L'organigramme de LETTRE est alors modifié comme l'indique la figure VI-7. (Remarque : certaines modifications sont en rapport avec les fonctions FNα , elles seront étudiées dans un paragraphe spécial).


LETTRE
A205
((PTR))='l'
oui
non
A230
(AN)='E'
oui
non
A240
(IND1) ?
=1
A270
=0
A260
((P)-1)=<110>
<109> → ((P)-1)
((PTR)) → (NF')
(IND14) ?
≠'10'
='10'
A262
C560
A261
(IND14)∧'01' → (IND14)
Cars
(PTR)+1 → (PTR)
A245
((PTR))='('
oui
non
A250
((PTR))='ch'
oui
non
((PTR))='b'
oui
non
A265
<132> → ((P))
(P)+1 → (P)
0 → ((P))
0 → (AIG)
(AN)16 → (CALC11)
(AN)10 → (CALC12)
'b' → (CALC2)
(IND14) ?
≠0
=0
C500
Cars
A275
<134> → ((P))
(P)+1 → (P)
0 → ((P))
(AN)16 → (CALC11)
(AN)10 → (CALC12)
Cars
(AN)='F'
non
oui
((PTR))='N'
non
oui
<110> → ((P))
(P)+1 → (P)
(IND14)∨'10' → (IND14)
((PTR)) → (NP')
(PTR)+1 → (PTR)
Cars
A210

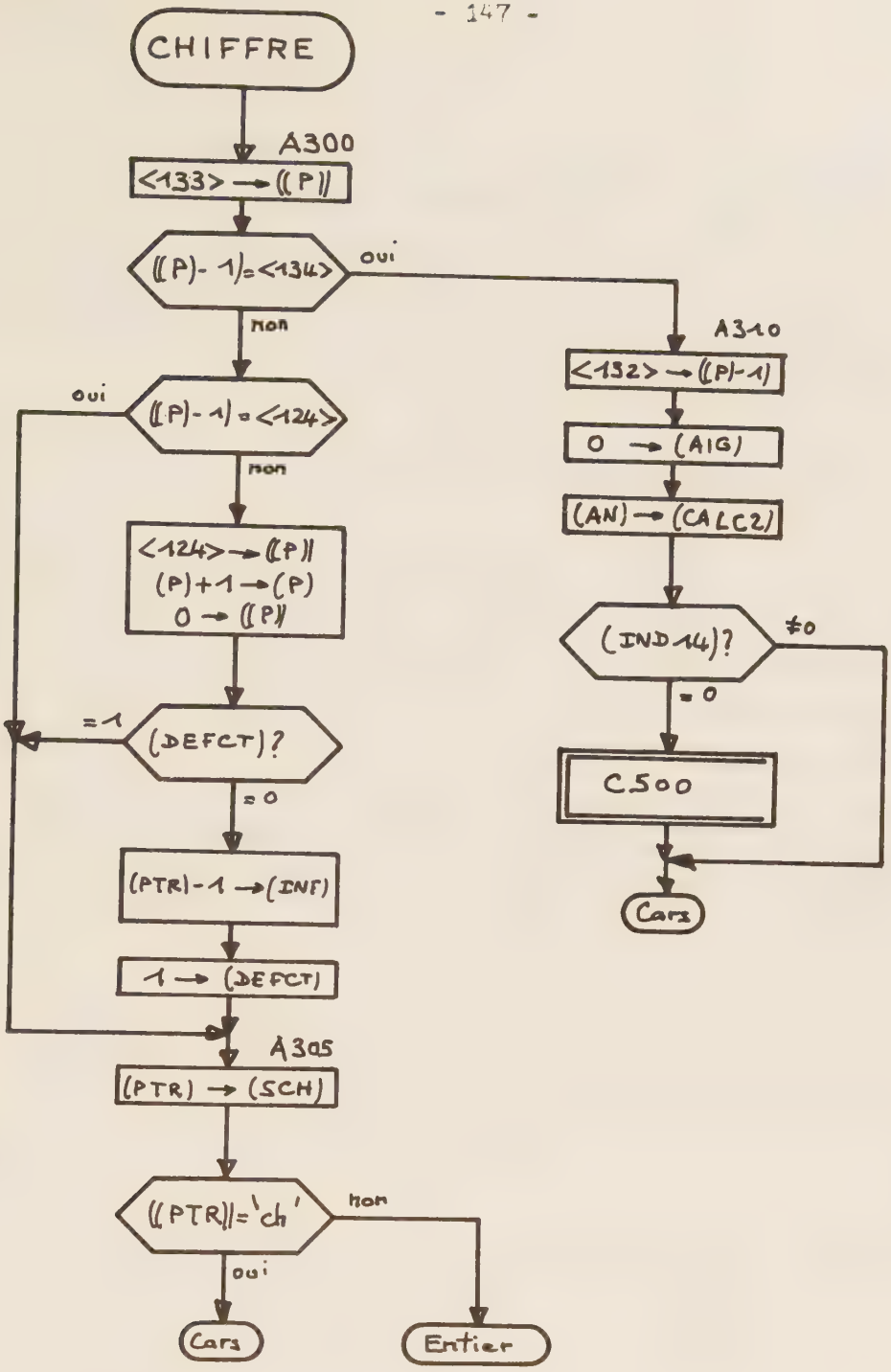

CHIFFRE
A300
<133> → ((P))
((P)-1)=<134>
oui
non
A310
<132> → ((P)-1)
0 → (AIG)
(AN) → (CALC2)
(IND14)?
≠0
=0
C500
Cars
((P)-1) = <124>
oui
non
<124> → ((P))
(P)+1 → (P)
0 → ((P))
(DEFCT)?
=1
=0
(PTR)-1 → (INF)
1 → (DEFCT)
A305
(PTR) → (SCH)
((PTR))='ch'
non
oui
Cars
Entier

Figure VI-8

L'organigramme de CHIFFRE est également modifié, puisque < variable simple > est un successeur de < chiffre > dans le graphe d'ordre partiel. Cette modification n'intervient qu'à la reconnaissance du code < 134 > dans la 2è cellule de la pile P. La figure VI-8 montre le traitement de variable simple, repérée A310.

VI-4 - PILE D'INTERPRETATION - PILE AUXILIAIRE

Le fait que le travail de compilation se fait en mode interprétatif, impose d'avoir constamment à disposition, les opérandes sur lesquels on veut effectuer des opérations. Un moyen commode de résoudre ce problème, consiste à utiliser une pile, dans laquelle, on stocke les adresses réelles des opérandes numériques utilisés, ainsi que dans certains cas, le symbole de l'opération à réaliser. Le résultat d'une opération sera mis dans une pile auxiliaire (appelée AUX), dans une cellule dont on stockera l'adresse dans la pile d'interprétation (appelée T).

Soit par exemple à réaliser l'opération 22 + 15 =

La pile T se présente sous la forme :

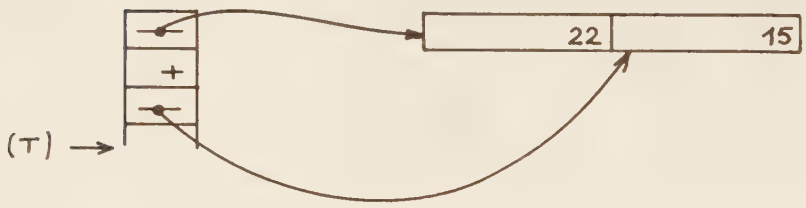


Figure VI-9

On réalise l'opération ((T)-3) o ((T)-2) o ((T)-1) ⟶ ((AUX))

Après l'opération, on a :

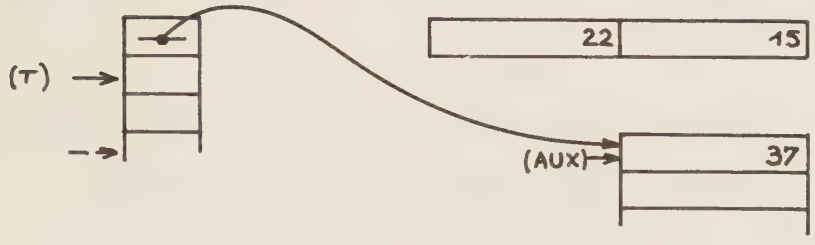


Figure VI-10

Une fois l'opération réalisée, on peut réutiliser les 3 cellules de la pile interprétation, en fait, on recule le pointeur T de 2 crans et on stocke l'adresse du résultat, qui est dans ((AUX)), dans ((T)-1).

Remarquons que les opérandes sur lesquels on travaille, peuvent être déjà logés dans la pile AUX, ou dans la pile des valeurs numériques des variables simples par exemple.

La pile interprétation, joue un grand rôle dans tous les modules d'interprétation, son évolution continuelle permet d'avoir constamment accès à toutes les quantités voulues, ceci quel que soit le type de l'instruction prise en charge.

Nous verrons mieux l'utilisation de la pile T, lors de l'exposé du traitement de chaque instruction BASIC.

Remarquons que la pile AUX est une pile descendante pour des raisons pratiques dues à l'encombrement en MC.

VI-5 - TRAITEMENT DES QUANTITES NUMERIQUES

On appelle quantité numérique, toute chaîne terminale dont l'analyse se termine par l'élément syntaxique < nombre > , ou par l'élément syntaxique < constante > (dans le cas d'une quantité numérique appartenant à un DATA).

Il y a une différence fondamentale entre les variables et les quantités numériques : la valeur numérique affectée à une variable est une information qu'il faut conserver tant que la variable est utilisée, a priori entre l'instant où l'on commande l'affectation et l'instant de la fin d'exécution du programme, c'est la raison d'être de la pile des valeurs numériques des variables ; une quantité numérique est au contraire une information éphémère qui existe en format interne flottant entre la fin de conversion forme externe ——→ forme interne et la fin d'exécution de l'instruction en cours.

La pile AUX accueille tout naturellement les quantités numériques du type <nombre> dans des cellules dont les adresses sont empilées dans T.

Le problème est différent pour les constantes, celles-ci étant affectées à des variables, c'est alors la pile des valeurs numériques des variables simples qui reçoit les résultats de conversion dans le cas des variables simples, ou bien l'élément du tableau considéré, dans le cas des variables indicées. Le traitement des constantes sera examiné en détail avec les ordres READ et INPUT.

Le programme de conversion, INTFLØ est capable de traiter toutes les formes externes possibles de nombres réels. Il demande

avant conversion, l'adresse du premier caractère constituant la chaîne numérique. Cette adresse qu'on appelle INF, est connue dans le module CHIFFRE (figure VI-8), la première fois qu'on détecte un chiffre, l'indicateur DEFCT étant alors à 0, ou bien dans le module POINT (figure VI-11).

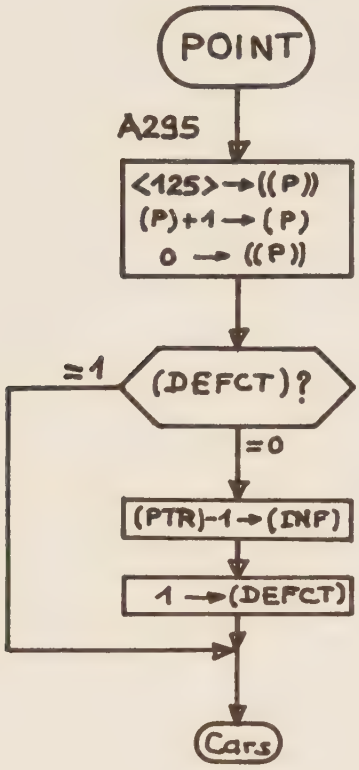


Figure VI-11

L'indicateur DEFCT normalement à zéro, passe à 1 à la détection du premier caractère d'une chaîne terminale dont l'analyse indique que sa structure est celle d'un nombre. DEFCT est ramené à 0 à la fin du traitement de l'élément < nombre > après la conversion en flottant interne. La figure VI-12 représente le traitement de < nombre > .

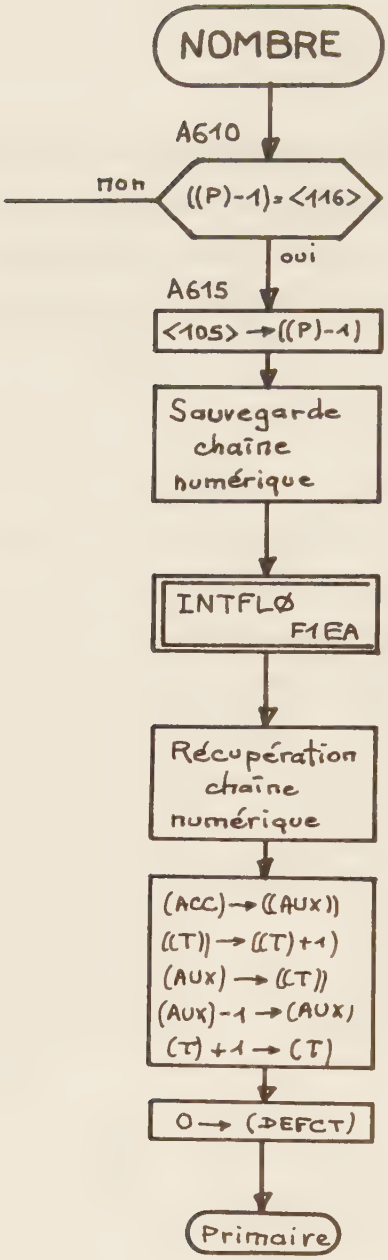


Figure VI-12

Cet organigramme appelle quelques remarques. Le sous-programme de conversion est compris entre une séquence de sauvegarde et une séquence de récupération de la chaîne numérique, en effet, INTFLØ demande que la chaîne à convertir soit close par une virgule, qu'il faut mettre à la place du premier caractère ne faisant plus partie de la structure < nombre > , d'autre part, pour les besoins de la conversion,

le sous-programme modifie physiquement la chaîne d'entrée, celle-ci faisant partie d'une instruction du programme qu'il est hors de question de modifier, d'où les 2 séquences.

On peut remarquer qu'on décale d'un cran le contenu de la pile T avant d'y mettre l'adresse de la cellule de la pile AUX contenant le résultat de la conversion.

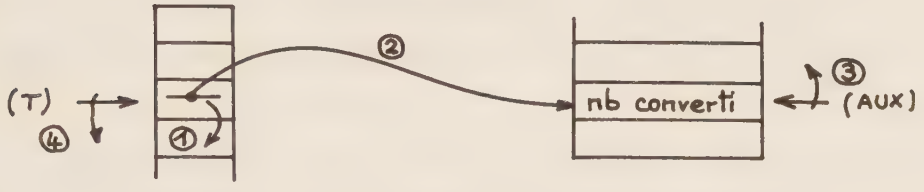


Figure VI-13

L'opération ① s'explique par le fait qu'il peut y avoir dans la cellule courante de T, un symbole d'opération jouant le rôle de délimiteur entre le nombre qu'on considère (et qui se trouve être le premier opérande) et le second opérande. Ce symbole d'opération est introduit en ((T)) à sa détection, il ne commande pas l'évolution de (T) en (T)+1, il permet par l'aiguillage de terminer l'examen de la structure précédente, ici $<$nombre$>$. Il est donc important de ne pas perdre le contenu de la cellule courante de T. L'opération ② est banale, son corollaire étant les opérations ③ et ④ (la pile AUX, on le rappelle, évolue en sens inverse de la pile T).

VI-6 - TRAITEMENT DES VARIABLES INDICEES ET DES MATRICES

VI-6-1 - Dimensions

Rappelons qu'on appelle matrice, tout tableau de coefficients ayant P lignes et Q colonnes. En BASIC, il est possible de traiter des tableaux à une et à deux dimensions. En fait, il est plus simple (pour avoir un traitement unique des 2 cas) de considérer un tableau à une dimension comme un tableau à 2 dimensions, dont la seconde égale 1, c'est alors une matrice-colonne. La matrice-ligne, quant à elle, requiert l'écriture en clair des 2 dimensions (la 1ère étant évidemment égale à 1).

Nous imposons dans cette étude que toute matrice soit déclarée dans un ordre DIM (dimension).

Les dimensions citées dans l'ordre DIM sont les dimensions maximales que peut prendre la matrice au cours du programme, il importe donc de connaître tout au long du programme les dimensions courantes et les dimensions maximales de toutes les matrices.

VI-6-2 - Encombrement

Chaque élément d'une matrice occupe 5 octets, comme une variable simple. L'encombrement de toutes les matrices réunies devient prohibitif assez facilement, il ne peut donc pas être question de conserver les matrices en mémoire centrale.

La solution proposée consiste à stocker les matrices sur le tambour, et comme le tambour est assez vaste, on prendra une piste par matrice, ce qui autorise un nombre maximal d'éléments égal à 700/5, soit 140 éléments par matrice.

Si P est la 1ère dimension citée dans un ordre DIM et Q la 2ème, il faut que la relation suivante soit vérifiée : $P * Q \leqslant 140$.

VI-6-3 - Identificateurs de matrice

Le problème se pose de retrouver la matrice en connaissant son nom, il faut alors trouver le numéro de la piste qui contient les éléments. On y arrive en utilisant les identificateurs de matrice :

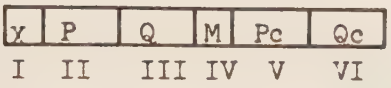


Figure VI-14

I : Nom de la matrice
II, III : Dimensions citées dans DIM (lignes, puis colonnes)
IV : Numéro de la piste affectée à la matrice
V, VI : Dimensions courantes (lignes, puis colonnes)

On stockera l'ensemble des identificateurs de matrice, au bout de la piste qui contient le bloc de renseignements sur les DATA.

On reconnait qu'on a affaire à un identificateur de tableau dans LETTRE en A240, quand on reconnait que l'indicateur IND1 est à 1, ce qui indique un contexte matriciel.

Il s'agit de savoir si l'on définit une matrice, ou si l'on en utilise une, le traitement n'étant pas le même, dans le premier cas, on crée un identificateur de matrice, dans le second, on cherche à retrouver le bon identificateur, la séparation entre les 2 cas se fait grâce à l'indicateur IND10 qui est à 1 dans le cas d'ordre DIM. L'organigramme suivant fait suite à la sortie A270 de l'organigramme de la figure VI-7.

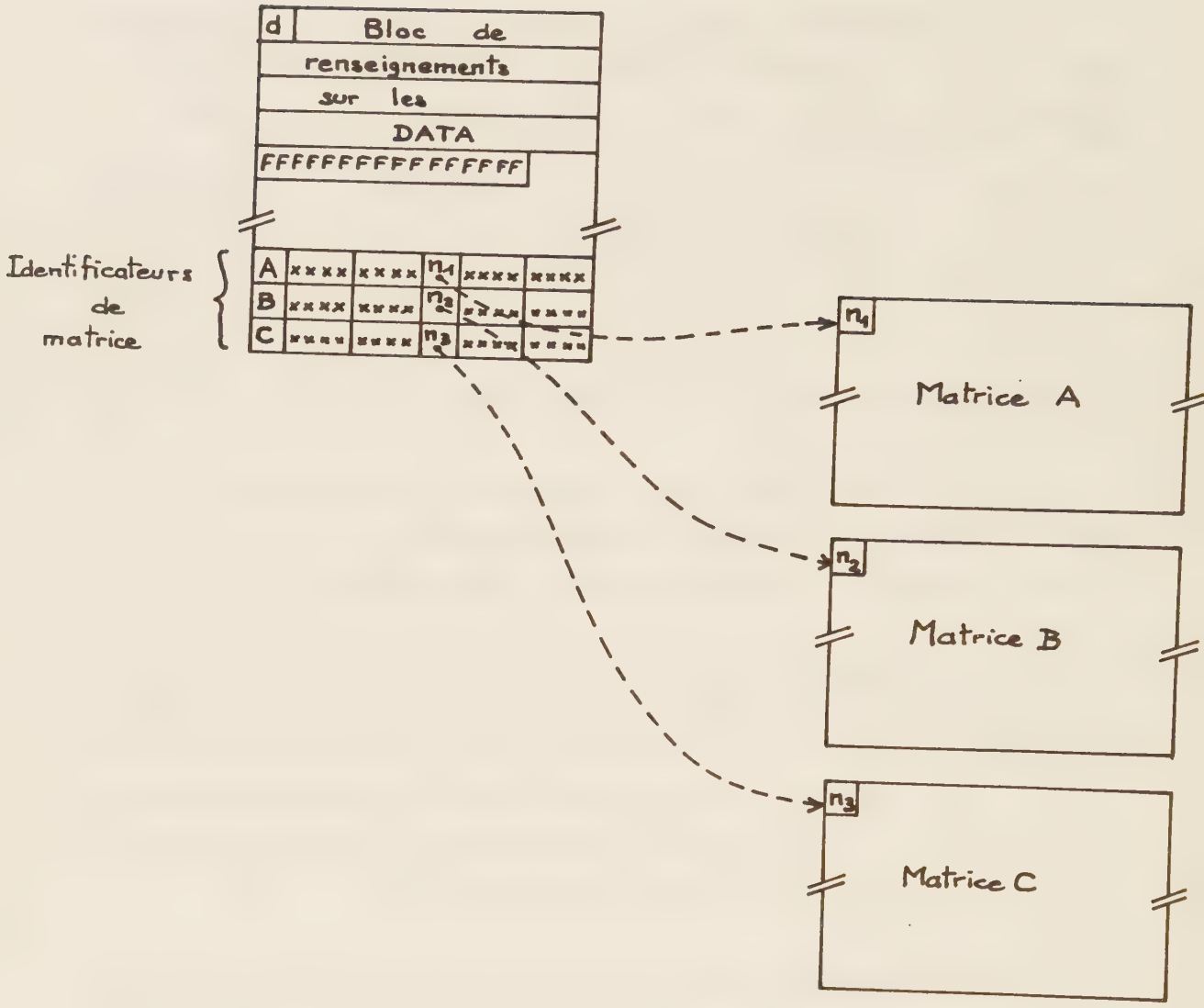

d
Bloc de
renseignements
sur les
DATA
FFFFFFFFFFFFFFFF
Identificateurs
de
matrice
A
B
C
Matrice A
Matrice B
Matrice C

Figure VI-15

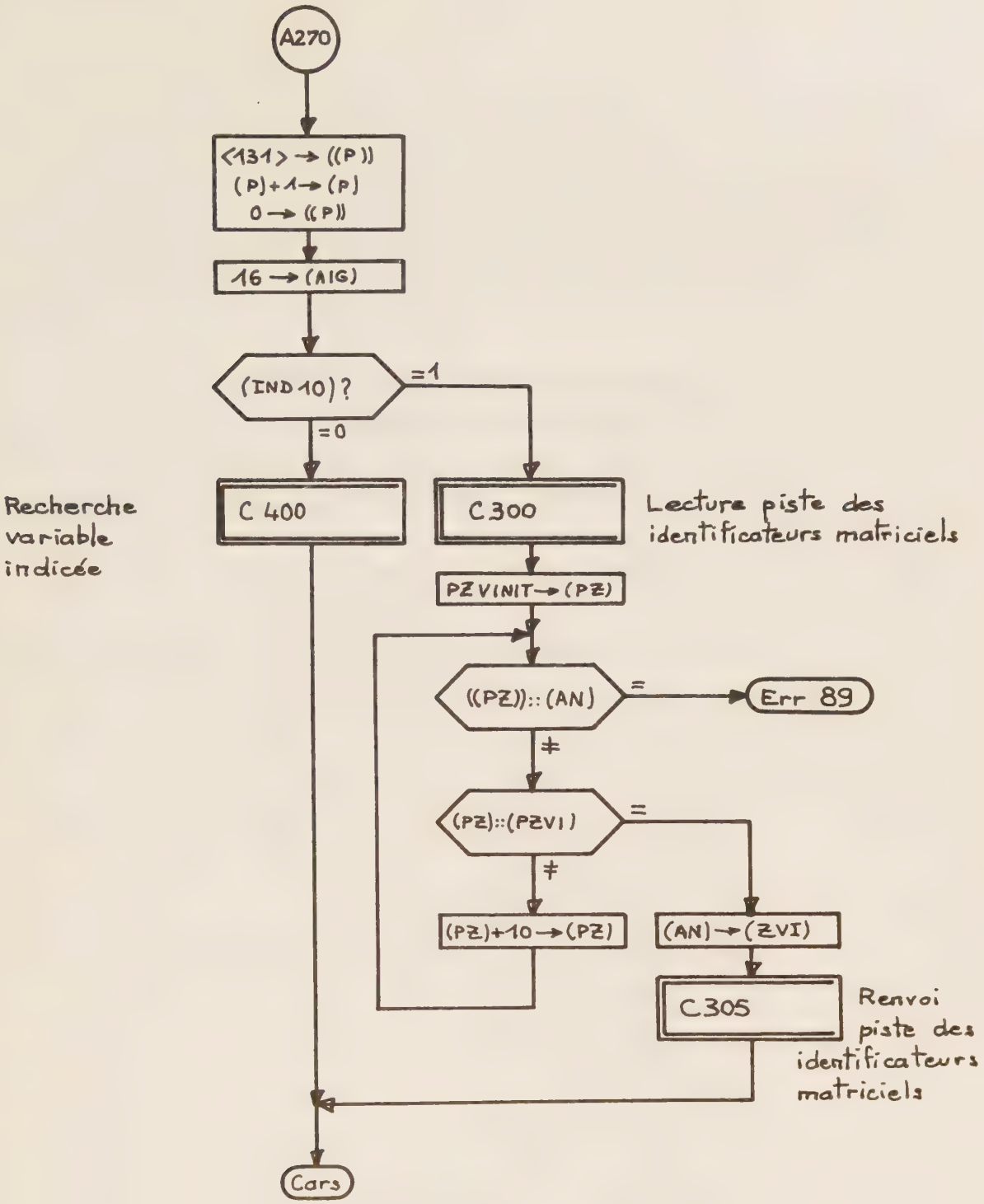

A270
<131> → ((P))
(P)+1 → (P)
0 → ((P))
16 → (AIG)
(IND 10)?
=1
=0
Recherche variable indicée
C 400
C 300
Lecture piste des identificateurs matriciels
PZVINIT → (PZ)
((PZ)):: (AN)
=
Err 89
≠
(PZ)::(PZVI)
=
≠
(PZ)+10 → (PZ)
(AN) → (ZVI)
C 305
Renvoi piste des identificateurs matriciels
Cars

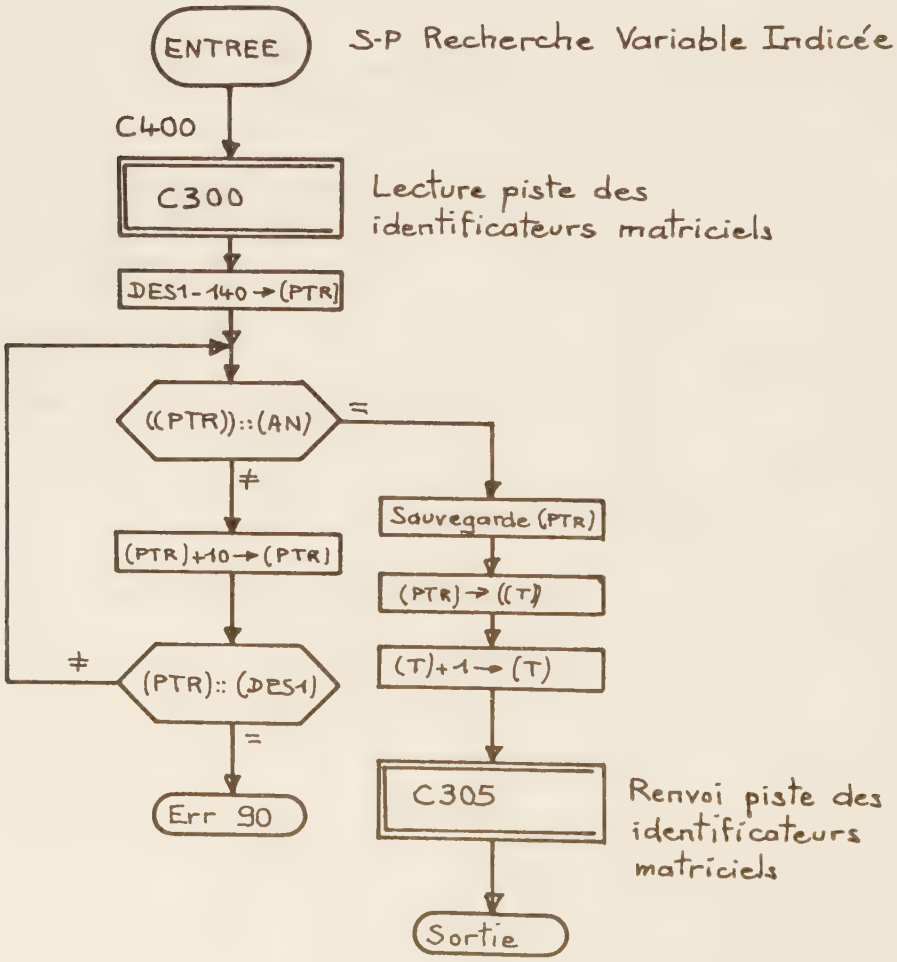
Figure VI-16

Il est souhaitable de vérifier si une matrice n'a pas déjà été définie, quand on rencontre un ordre DIM, car il peut y avoir conflit entre les 2 déclarations. En balayant la table des identificateurs matriciels depuis l'origine jusqu'à la 1ère cellule libre, on s'assure qu'il n'y a pas double définition, sinon on indique une erreur.

S'il n'y a pas double définition, on introduit le nom de la matrice dans la 1ère zone de l'identificateur.

L'introduction du numéro de la piste associée à la matrice, se fait en même temps que la mise en place des dimensions maximales (voir ce point avec le traitement de l'ordre DIM).

En phase utilisation des identificateurs de tableau, IND10 est à 0, on fait appel au sous-programme C400.



Pour trouver l'identificateur matriciel voulu, on balaye la table depuis l'origine, on compare le nom inscrit dans la 1ère zone des identificateurs avec le nom courant contenu dans la position d'analyse AN. Quand on a trouvé le bon identificateur, on insère son adresse dans la pile d'interprétation, ce qui permet quand on a besoin de la matrice d'obtenir très rapidement les renseignements voulus.

Une autre possibilité d'exploitation des identificateurs matriciels existe au niveau des variables indicées, dont on reconnait la structure dans OUVPAR par la filière A200. On doit faire appel au S/P C400.

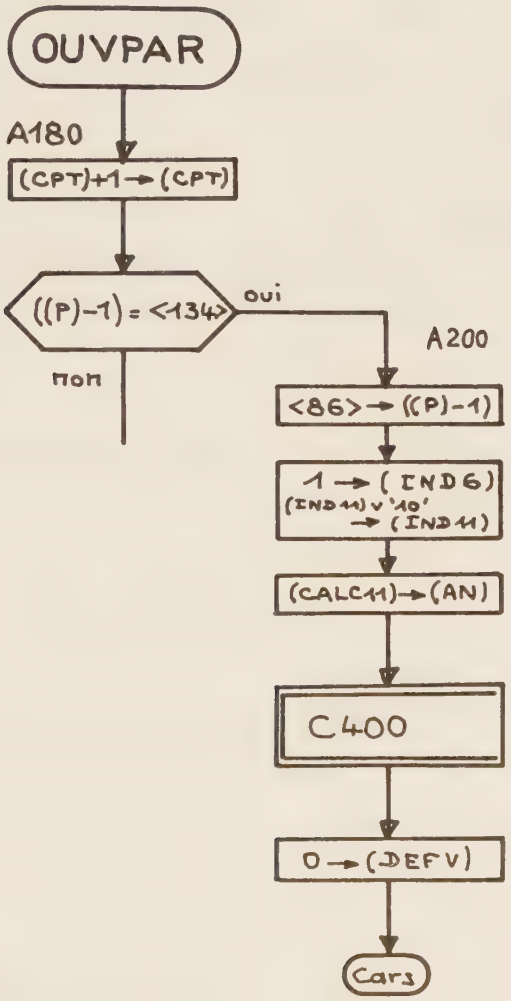


Figure VI-18

Avant d'appeler C400, il convient de mettre dans (AN), le nom de la matrice dont on utilise un élément, en effet, dans OUTPAR , AN contient une parenthèse d'ouverture, on récupère dans CALC11 le nom dont on a besoin.

La remise à zéro de DEFV est faite pour éviter de considérer les variables simples qui jouent le rôle d'indices, comme étant en état de définition.

VI-6-4 - Utilisation des variables indicées dans les expressions arithmétiques

Dans le corps d'expressions arithmétiques, l'évaluation des variables indicées pose certains problèmes, particulièrement quand il s'agit de structures emboitées.

Soit par exemple, l'expression : I + J * A (I, B (I, J)), au moment de l'évaluation des indices de l'élément de la matrice A, on se trouve amené à prélever la valeur de l'élément d'indices I et J de la matrice B.

En utilisant la pile T pour stocker les adresses des indices, on se trouve devant la situation suivante :

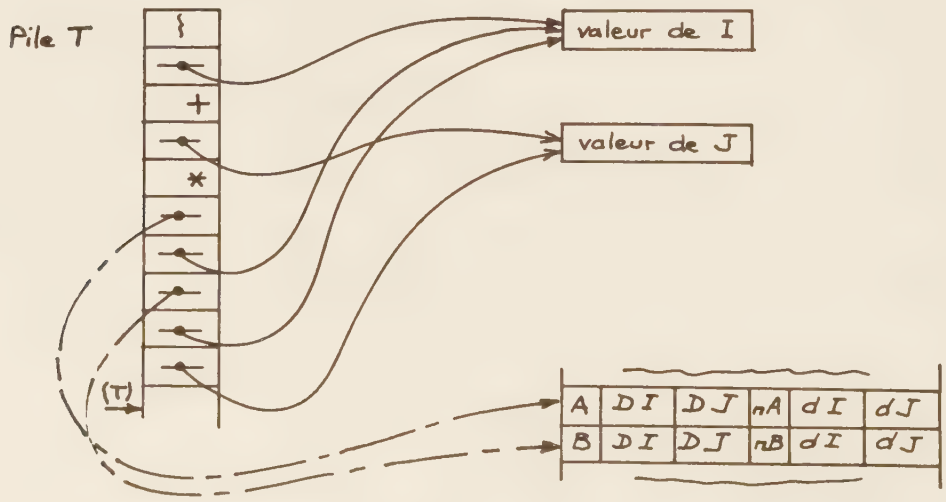


Figure VI-19

Il faut lire la matrice B, prélever l'élément B (I, J), l'introduire dans la pile AUX en fournissant sa nouvelle adresse à la pile T, de façon à permettre la recherche de l'élément voulu de la matrice A.

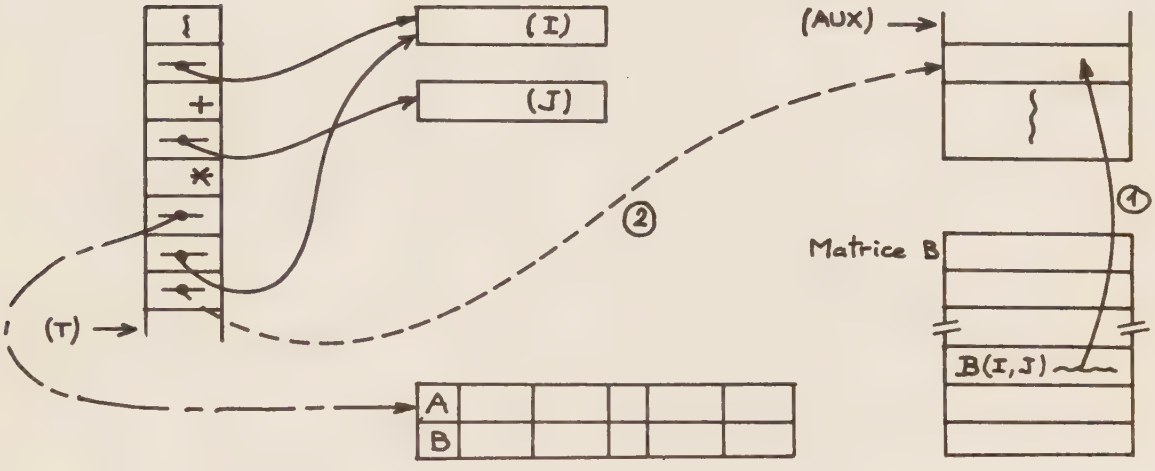


Figure VI-20

Ce dernier traitement est exécuté au niveau de la routine LISTE D'INDICES , ainsi qu'un certain nombre de vérifications.

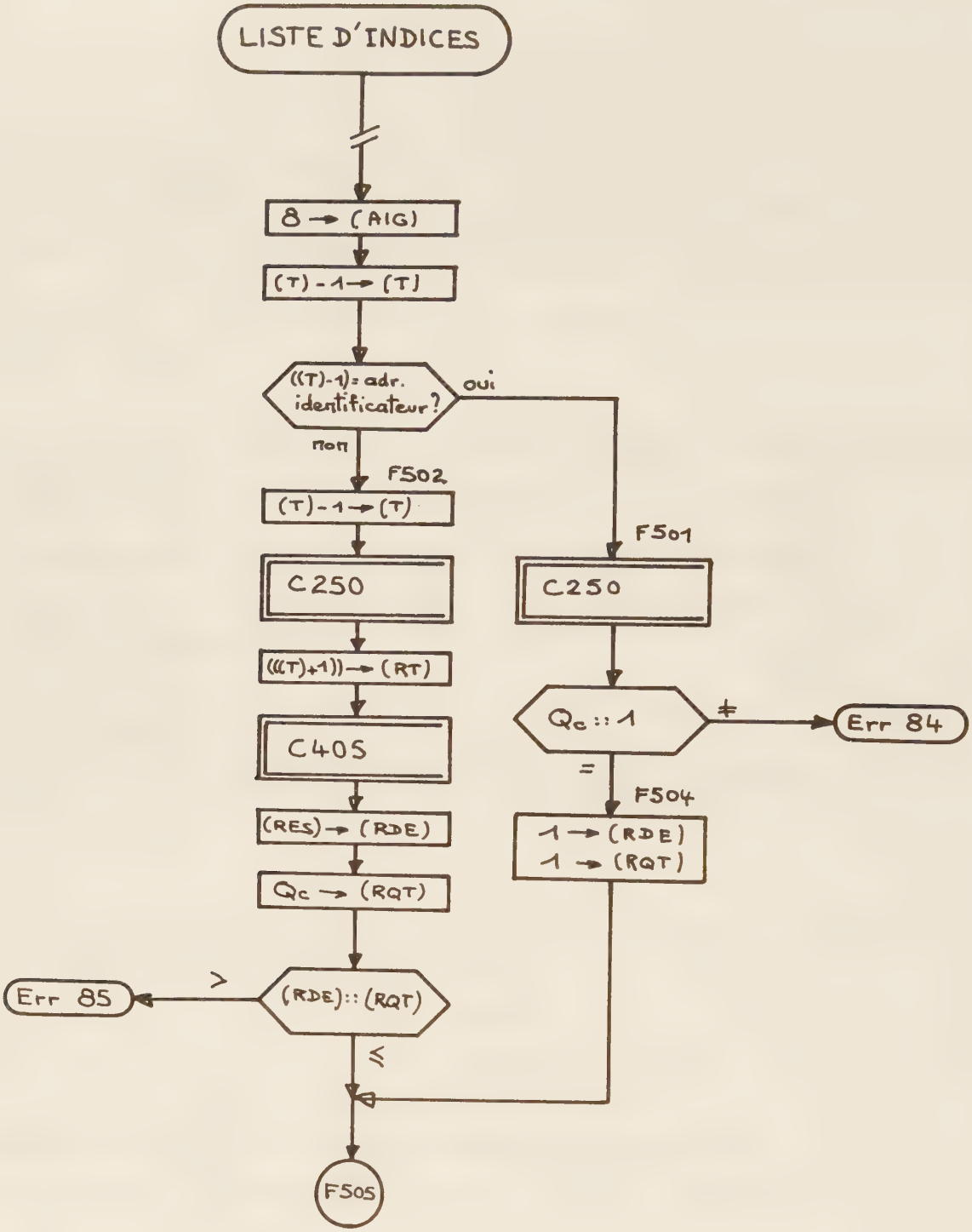

LISTE D'INDICES
8 → (AIG)
(T) - 1 → (T)
((T)-1) = adr. identificateur ?
oui
non
F502
(T) - 1 → (T)
C250
((T)+1)) → (RT)
C405
(RES) → (RDE)
Qc → (RQT)
(RDE) :: (RQT)
>
Err 85
≤
F501
C250
Qc :: 1
≠
Err 84
=
F504
1 → (RDE)
1 → (RQT)
F505

Figure VI-21

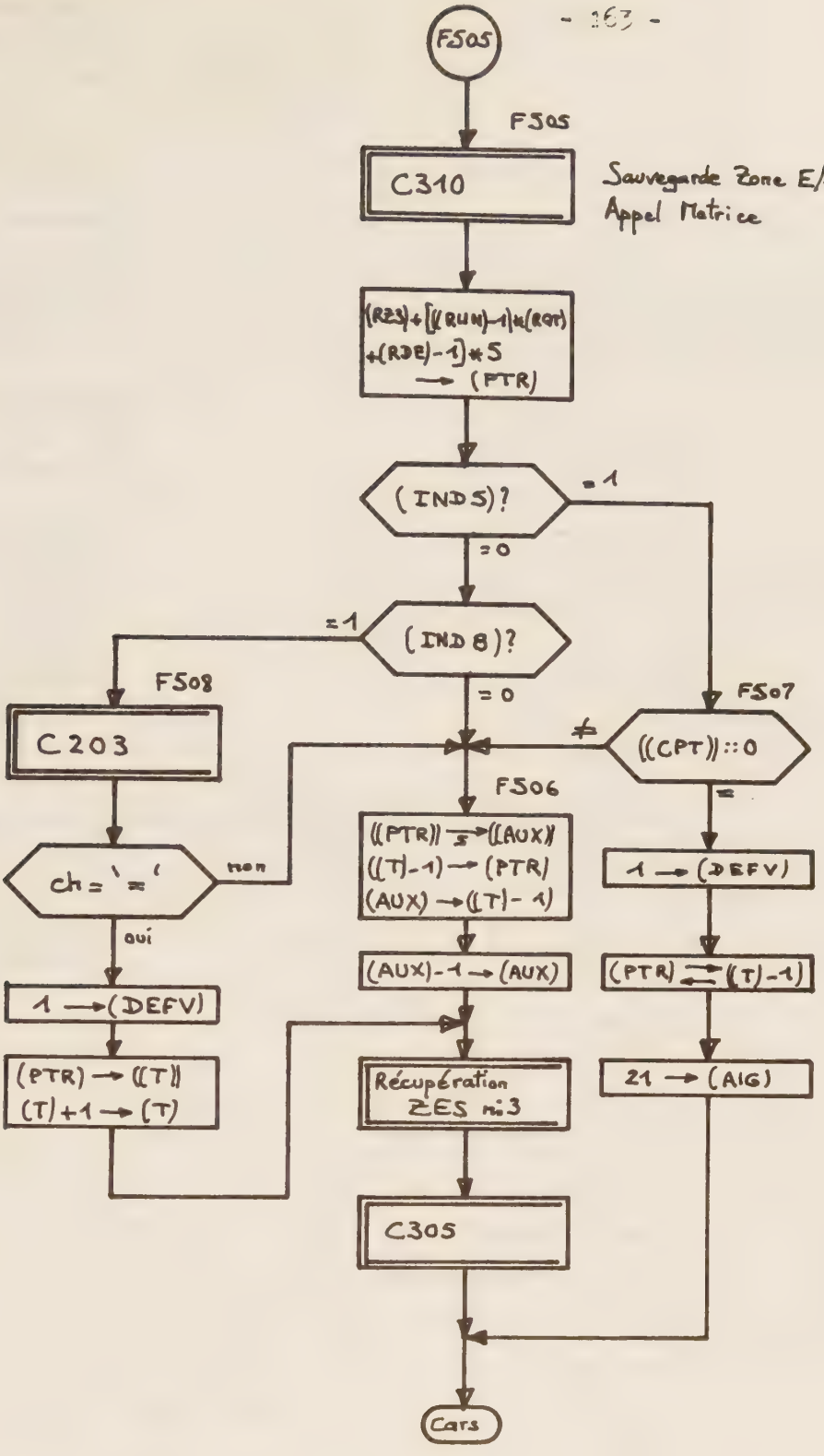

FS05
FS05
C310
Sauvegarde Zone E/S ni 3
Appel Matrice
((R23)+[((RUN)-1)*(RQT)
+(RDE)-1]*5
→ (PTR)
(IND5)?
=1
=0
(IND8)?
=1
=0
FS08
C203
FS07
≠
((CPT))::0
=
FS06
ch = ' ' ?
non
oui
((PTR)) →((AUX))
((T)-1) → (PTR)
(AUX) → ((T)-1)
1 → (DEFV)
(AUX)-1 → (AUX)
(PTR) ⇄ (T)-1
1 → (DEFV)
(PTR) → ((T))
(T)+1 → (T)
Récupération
ZES ni3
21 → (AIG)
C305
Cars

Figure VI-22

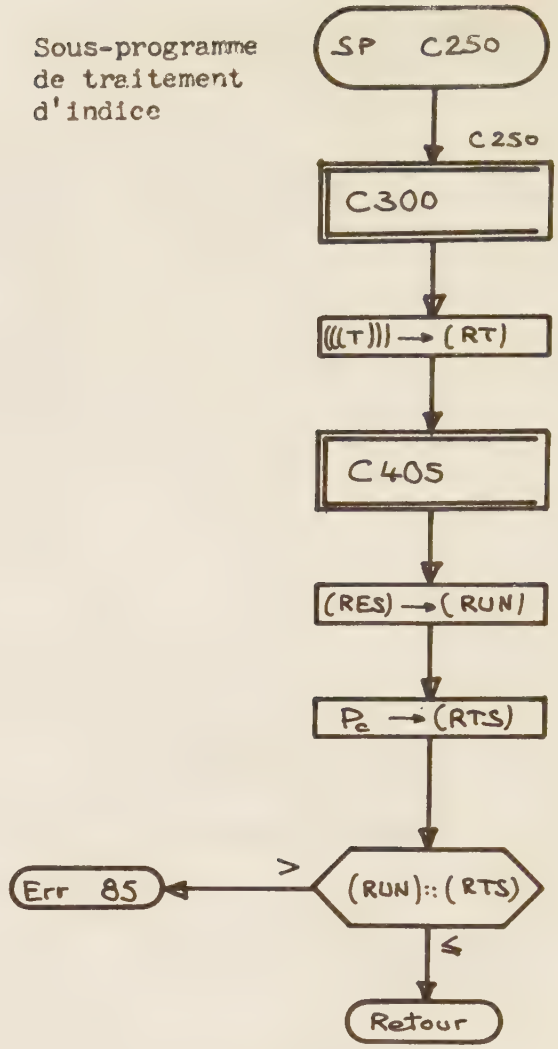


Figure VI-23

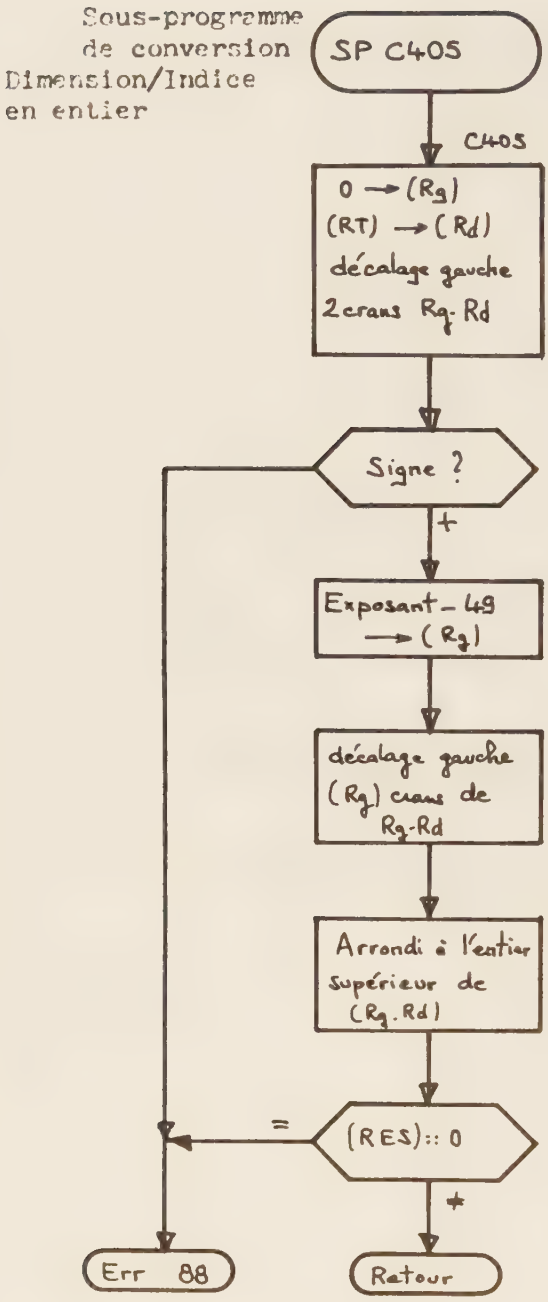


Figure VI-24

Quand on entre dans LISTE D'INDICES , on est sûr que la cellule d'adresse (T)-1, de la pile d'interprétation, contient l'adresse d'un indice, qui est unique dans le cas d'un tableau unidimensionnel, il peut y avoir un autre indice dont on cherche l'existence (c'est en fait le premier) :

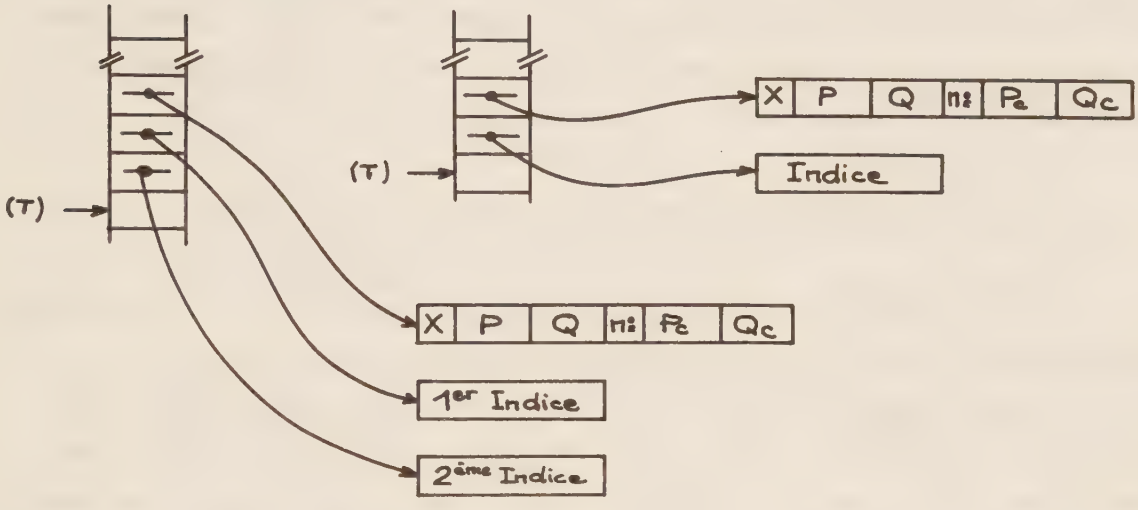


Figure VI-25

Après avoir reculé T d'un cran, si le tableau n'a qu'une dimension, on trouve dans ((T)-1) l'adresse de l'identificateur, on emprunte alors le chemin F501, où l'on convertit l'indice en entier, puis on vérifie que le tableau n'a bien qu'une dimension. Dans le cas où l'on a 2 indices, on traite le second de la même façon que le premier en vérifiant (voir C405, figure VI-24) que l'indice est strictement positif et que sa valeur est inférieure ou égale à la dimension courante (voir C250 figure VI-23).

Une fois les indices convertis en entiers, on emprunte le chemin F505 dans lequel on commande la lecture de la matrice, puis on calcule l'adresse de l'élément d'indices I et J par la formule suivante : $\left[(I-1) * Q_C + (J-1)\right] * 5 +$ Ad (1er élément), les éléments étant rangés ligne à ligne dans l'enregistrement.

En utilisation normale d'un élément de matrice dans le corps d'une expression arithmétique, les indicateurs IND5 et IND8 sont à 0, la filière utilisée est donc F506, où l'on effectue le travail schématisé par la figure VI-20.

Les traitements qui correspondent aux indicateurs dans l'état 1, sont relatifs à des cas spécifiques des ordres LET, READ et INPUT, nous renvoyons donc à l'examen de ces ordres pour obtenir les renseignements manquants sur la figure VI-22.

VI-6-5 - Utilisation des identificateurs de tableau

On emploie les identificateurs de tableau dans les différentes instructions matricielles. Leur forme peut prendre deux aspects : ou bien l'identificateur est nu, auquel cas on utilise le tableau dans ses dimensions courantes, ou bien l'identificateur est suivi d'une ou deux dimensions écrites entre parenthèses, auquel cas ces dimensions deviennent les dimensions courantes. La pile T prend alors selon les cas les formes suivantes :

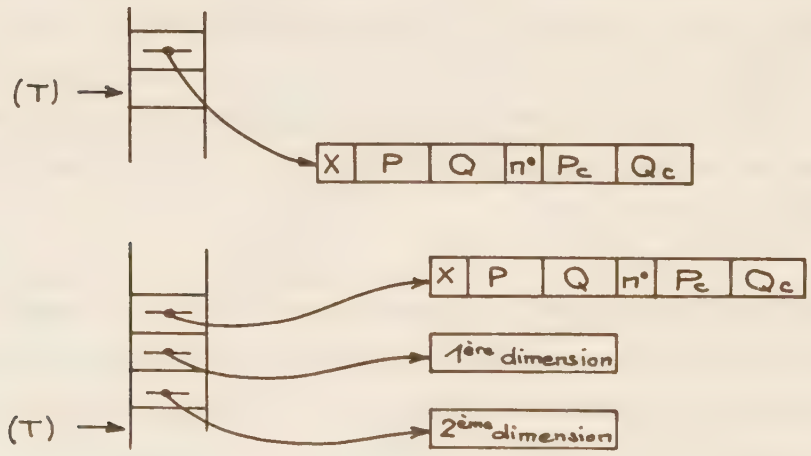


Figure VI-20

La mise en place de l'adresse de l'identificateur matriciel a déjà fait l'objet d'explications (voir S/P C400 - figure VI-17). Les dimensions quant à elles sont des expressions, et on les traite comme les indices. On introduit pour cela, un module "redimensionnement" dans la routine DIMENSIONS.

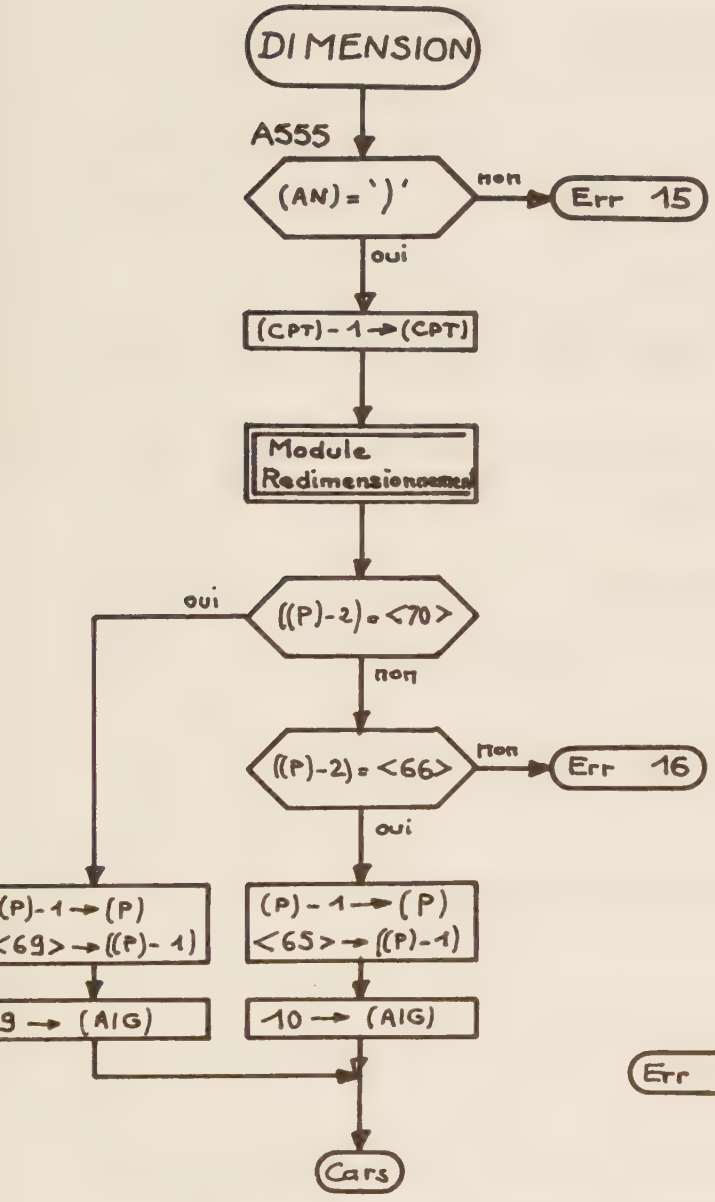


Figure VI-27

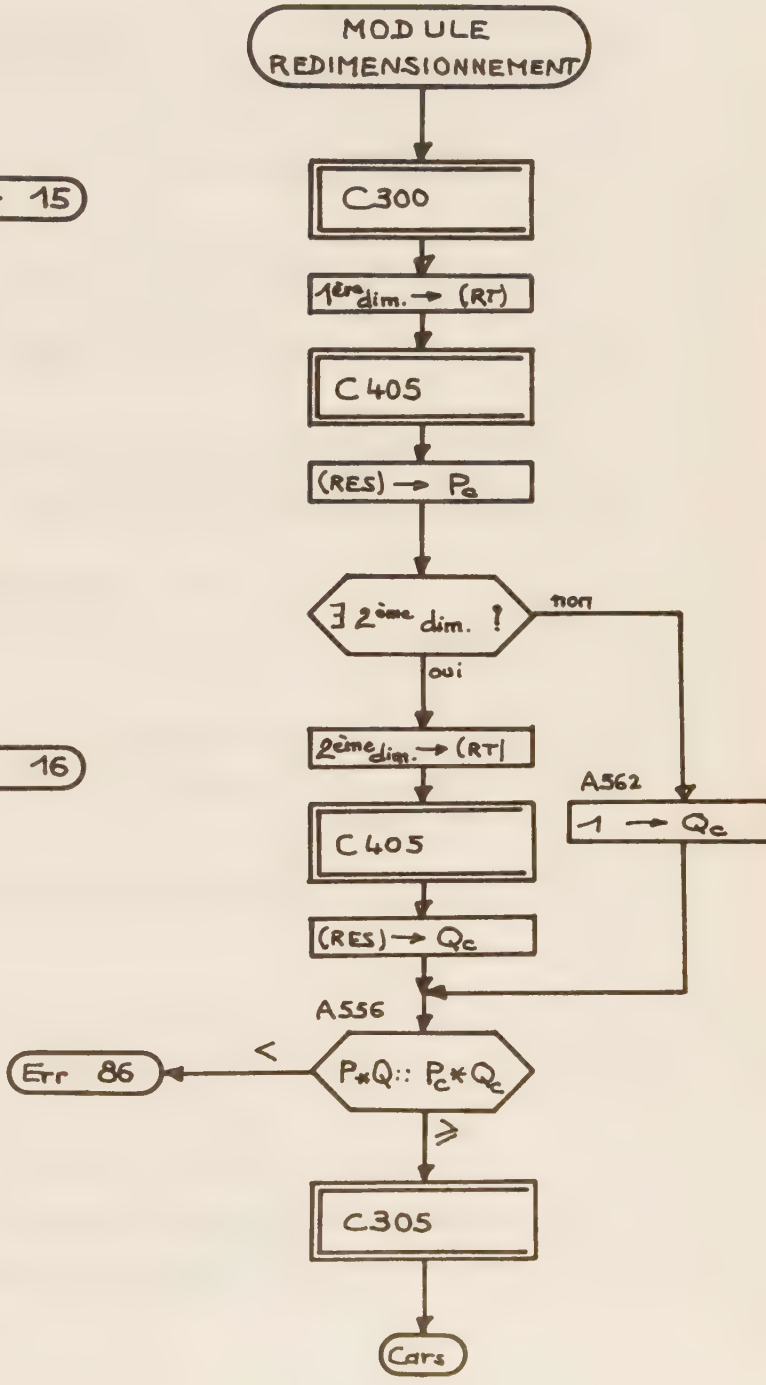


Figure VI-28

Après conversion, on vérifie que l'encombrement nouvellement défini du tableau, n'est pas supérieur à l'encombrement maximum autorisé par les quantités figurant dans l'ordre DIM (P et Q).

Les instructions matricielles telles qu'elles vont être exposées par la suite nous permettront de voir les points de détail relatifs aux identificateurs de tableaux.

VI-7 - TRAITEMENT DES FONCTIONS

VI-7-1 - Principes d'emploi

VI-7-1-1 - Fonctions prédéfinies

On emploie les fonctions prédéfinies dans le corps de toute expression arithmétique sous la forme :

NØM (argument)

Le nom de la fonction peut être l'un des suivants :

ABS : valeur absolue
ATN : Arc tangente
CØS : Cosinus
EXP : Exponentielle
LØG : Logarithme
SIN : Sinus
SQR : Racine carrée
TAH : Tangente hyperbolique.

L'argument est une expression arithmétique composée d'une suite quelconque de variables simples ou indicées, de nombres, de fonctions prédéfinies ou internes, séparés par des symboles d'opération.

VI-7-1-2 - Fonctions internes

Toute fonction interne doit faire l'objet d'une définition dans le cadre d'un ordre DEF, eg :

DEF FNA (α) = B + α + CØS (C/α)

FNA constitue le nom de la fonction (où A est une lettre quelconque)
α est le nom de la variable formelle qui doit être simple, cette variable formelle peut ne pas figurer dans le membre de droite (c'est alors une variable virtuelle).

L'utilisation d'une fonction interne s'effectue comme pour une fonction prédéfinie : FNA (argument).

L'argument est de même nature et il remplace en exécution la variable formelle.

Exemple : DEF FNC(X) = X ↑ 3
LET A = FNC(2)

L'argument 2 remplace la variable formelle X, on calcule donc 2 ↑ 3 = 8, valeur qu'on affecte à la variable A.

VI-7-2 - Solution proposée pour le traitement des fonctions internes

Etant donné qu'on a généralement besoin plusieurs fois de la même fonction, il faut conserver en mémoire le résultat de la compilation d'une définition de fonction. Ce résultat doit être directement exploitable par l'interpréteur, on est donc amené à intégrer des caractères spéciaux dans le corps des fonctions, pour différencier les variables formelles des variables normales du programme.

La forme proposée pour une fonction après compilation est la suivante :

Cellule corps de fonction

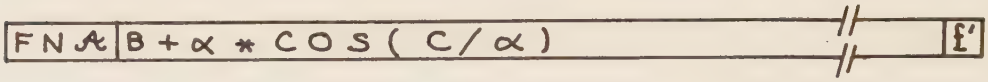


Figure VI-29

En exécution, on arrive à l'utilisation de FNA (argument), où l'on connait la valeur de l'argument, on recherche alors la cellule qui contient la fonction considérée, on lance la compilation sur le corps de la fonction, la détection du code "α ", indiquant qu'il faut travailler avec la valeur de l'argument, la détection du code "f'" de fin de corps de fonction indiquant qu'on a terminé et qu'il faut revenir à la compilation normale.

Sachant qu'on peut rencontrer d'autres fonctions à l'intérieur d'un corps de fonction, il faut prévoir le passage en compilation de ces fonctions avant l'achèvement de la compilation de la fonction courante. Il y a donc lieu de mettre en pile toutes les informations nécessaires à la reprise de compilation, en particulier, le caractère en analyse ainsi que le pointeur caractère dans l'instruction courante. En effet, quand on part en compilation sur un corps de fonction, on emploie l'algorithme d'interprétation, or ce dernier utilise la position d'analyse AN et le pointeur PTR qui vont évoluer, il faut donc restituer leurs valeurs quand on a terminé l'interprétation de la fonction.

Un autre problème soulevé par l'emploi des fonctions emboitées est celui de la transmission de l'argument. Soit par exemple, les

définitions suivantes :

```
DEF  FND (X) = P * X/180
DEF  FNC (X) = CØS (FND(X))
```

et soit la fonction FNC (A).

Quand on reconnait la fin de structure de la fonction FNC, on dispose dans la pile T, de l'adresse de l'argument A, mais en compilant FNC, il faut transmettre l'adresse à FND.

L'emploi d'une pile FØNCTIØN permet de résoudre tous ces problèmes, chaque cellule se présente sous la forme suivante :

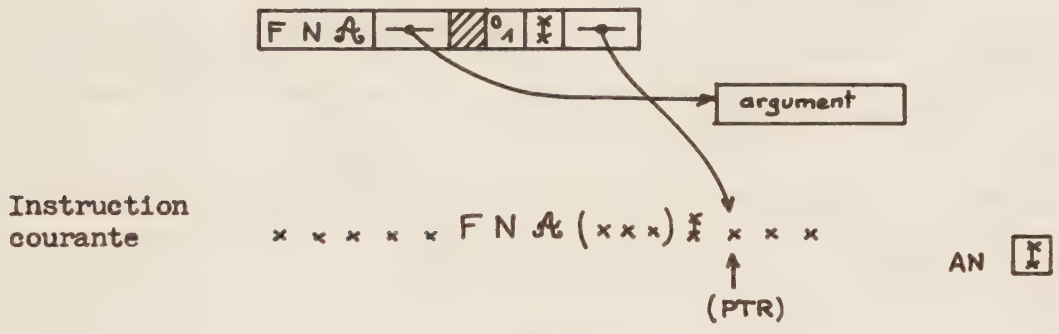


Figure VI-30

VI-7-3 - Organigrammes

VI-7-3-1 - Table des corps de fonction

On sait que la variable formelle est une variable simple, dans la routine VARIABLE SIMPLE on sépare les variables formelles grâce à IND14. Après la remise à zéro de IND14, on introduit le module qui permet d'introduire les caractères spéciaux et de créer la table qui est stockée sur le tambour.

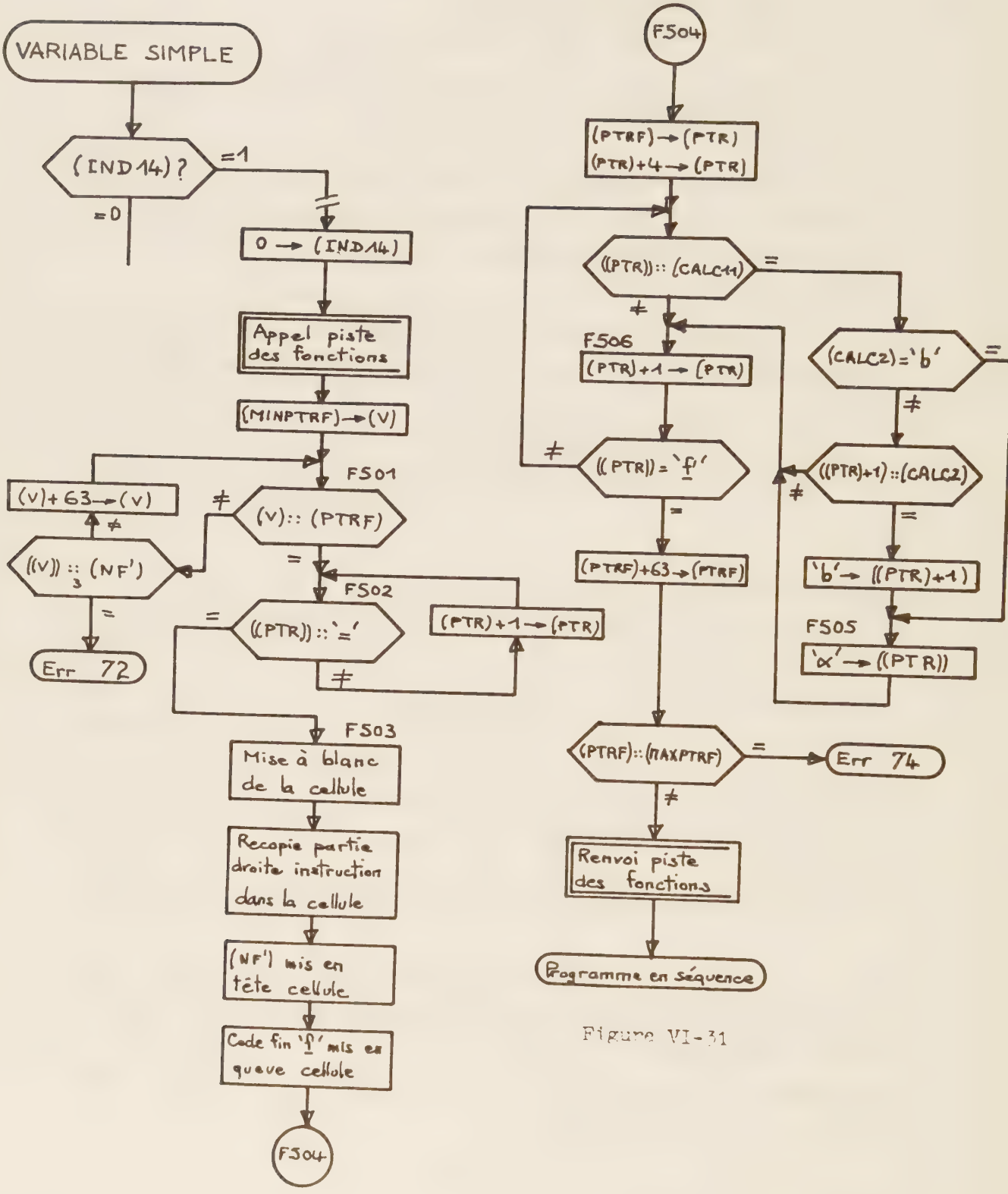

VARIABLE SIMPLE
(IND14) ?
=1
=0
0 → (IND14)
Appel piste des fonctions
(MINPTRF) → (V)
FS01
(V) + 63 → (V)
≠
(V) :: (PTRF)
((V)) :: 3 (NF')
=
Err 72
FS02
((PTR)) :: '='
(PTR) + 1 → (PTR)
FS03
Mise à blanc de la cellule
Recopie partie droite instruction dans la cellule
(NF') mis en tête cellule
Code fin 'f' mis en queue cellule
FS04
FS04
(PTRF) → (PTR)
(PTR) + 4 → (PTR)
((PTR)) :: (CALC1)
FS06
(PTR) + 1 → (PTR)
((PTR)) = 'f'
(PTRF) + 63 → (PTRF)
(PTRF) :: (MAXPTRF)
Err 74
Renvoi piste des fonctions
Programme en séquence
(CALC2) = 'b'
((PTR)+1) :: (CALC2)
'b' → ((PTR)+1)
FS05
'α' → ((PTR))

Figure VI-31

Cet organigramme appelle quelques commentaires. On appelle d'abord en MC la piste qui contient la table, en effet, sachant qu'on autorise 11 fonctions réduites à 63 caractères, cela fait une table de 693 octets qu'on peut loger sur une piste du tambour. On balaye alors la table pour vérifier qu'il n'y a pas double définition de la fonction (sinon, on indique ERR72), on cherche le signe égal dans l'instruction pour séparer la partie droite qu'on recopie dans la cellule de la table, en y ajoutant le nom de la fonction dans les 3 premières positions et le caractère spécial de fin de corps de fonction dans la dernière (63ème).

On dispose dans CALC11 et CALC2 du nom de la variable formelle qu'il faut remplacer par le caractère spécial "$\propto$ " : paramètre formel, dans le corps de la fonction.

On termine enfin, par la vérification du non-dépassement de la table et par le renvoi en tambour de la table.

L'instruction DEF ayant alors été complètement prise en charge, on amène l'instruction qui suit dans le programme BASIC, ce qui est réalisé par la routine "Programme en Séquence" qu'on verra plus en détail dans le paragraphe traitant de la prise en charge des instructions BASIC.

Remarquons qu'on peut prendre 63 des 75 caractères de l'instruction, en effet on peut s'affranchir du numéro de ligne (4 chiffres + 1 blanc), du mot-clé DEF (3 caractères), de la variable formelle entre parenthèses (au moins 3 caractères), et du signe égal (1 caractère).

VI-7-3-2 - Pile FØNCTION (PEF)

Cette pile sera constituée de cellules provenant indifféremment des fonctions internes et prédéfinies.

La mise en pile est faite par le module "Empilage Fonctions".

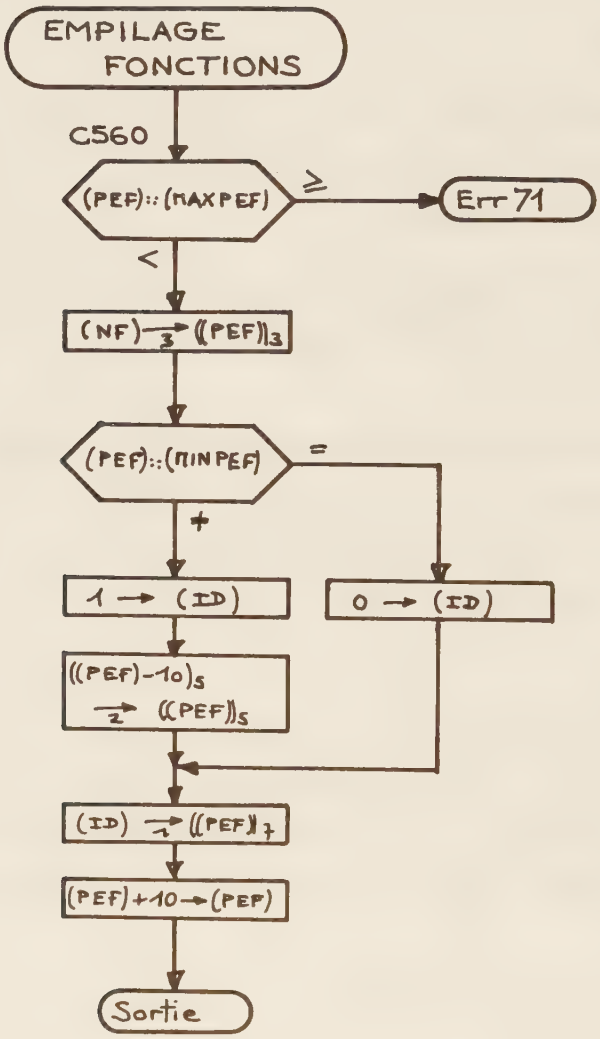


Figure VI-32

L'utilisation de fonctions récursives est interdite, aussi on peut vérifier que la pile ne dépasse pas un maximum raisonnable, pouvant constituer une barrière au-delà de laquelle on dira qu'il y a récursivité accidentelle (Err 71).

On introduit le nom de la fonction (prédéfinie ou interne, dans le 1er champ de la cellule courante, on positionne un indicateur à zéro, si la cellule est la première de la pile, à un dans le cas contraire, on recopie alors l'adresse de l'argument depuis la cellule précédente pour permettre l'emploi de fonctions emboîtées par les définitions, on enregistre l'indicateur, et on fait évoluer le pointeur.

Ce module va être utilisé à 2 endroits dans l'interpréteur, dans "lettre" (voir figure VI-7), au moment où l'on reconnait qu'on a affaire à une fonction (détection de <110> dans ((P)-1)), l'indicateur IND 14 permettant de séparer les différents cas qui se présentent pour les fonctions :

IND14 = '00'	{	cas normal
		après DEF FNA (α)
		après FNA sans DEF
'01'		après DEF FNA
'10'		après FN sans DEF
'11'		après DEF et FN

En utilisation d'une fonction, il n'y a pas le mot-clé DEF, il faut donc empiler une cellule à la détection de IND14 = '10', (appel de C560).

L'autre cas se présente avec les fonctions prédéfinies, ici, il suffit d'appeler le module à la détection des différentes fonctions :

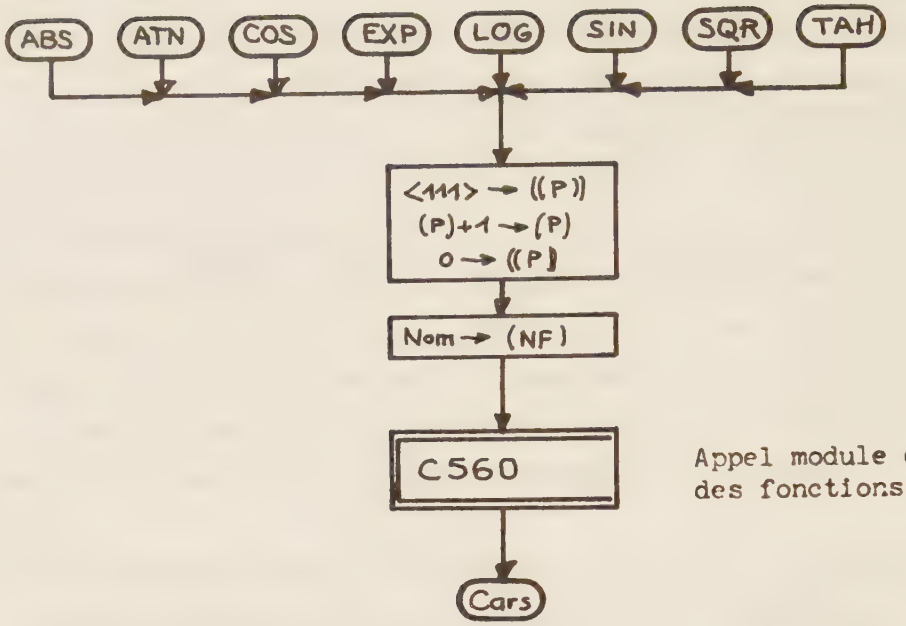


Figure VI-33

VI-7-3-3 - Argument - Compilation des fonctions

On sait que dans tous les cas, l'argument d'une fonction est une expression figurant entre parenthèses. C'est donc quand on repère cette dernière structure, qu'on peut interpréter les fonctions.

L'organigramme du traitement des expressions entre parenthèses devient alors :

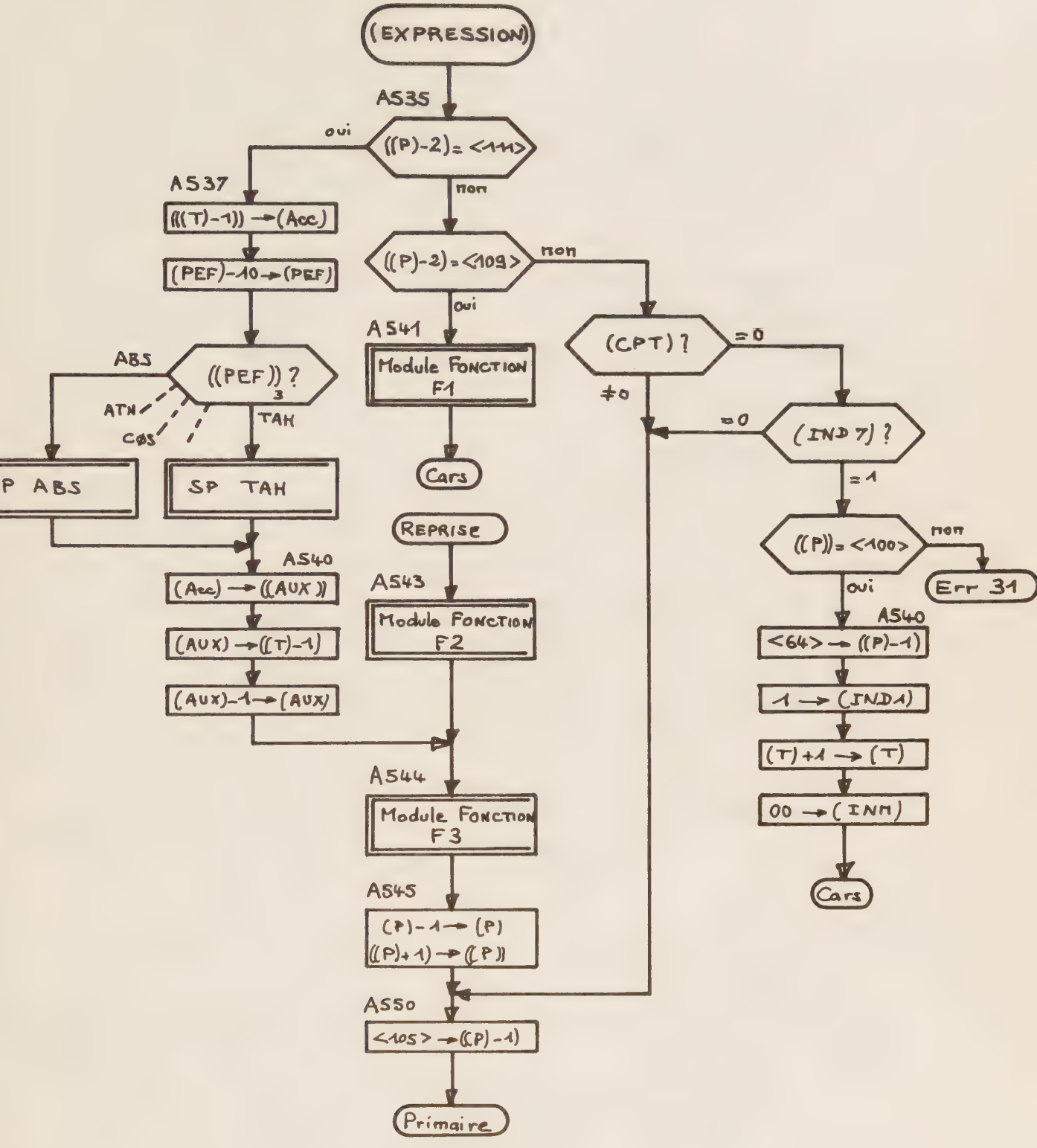

(EXPRESSION)
A535
((P)-2)= <111>
oui
non
A537
((T)-1)) → (Acc)
(PEF)-10 → (PEF)
((P)-2)= <109>
non
oui
A541
Module Fonction F1
(CPT) ?
=0
≠0
ABS
((PEF)) ?
3
ATN
CØS
TAH
P ABS
SP TAH
Cars
=0
(IND 7) ?
=1
REPRISE
A543
Module Fonction F2
((P))= <100>
non
oui
Err 31
A540
(Acc) → ((AUX))
(AUX) → ((T)-1)
(AUX)-1 → (AUX)
A540
<64> → ((P)-1)
1 → (IND1)
(T)+1 → (T)
00 → (INM)
Cars
A544
Module Fonction F3
A545
(P)-1 → (P)
((P)+1) → ((P))
A550
<105> → ((P)-1)
Primaire

Figure VI-34

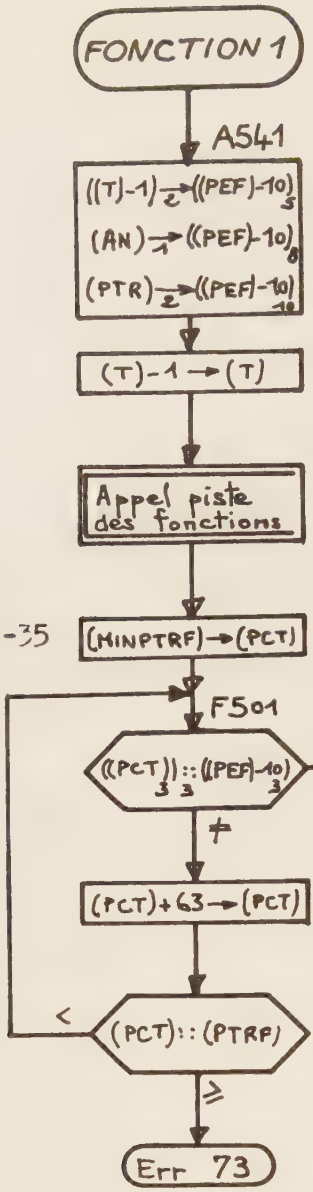


Figure VI-35

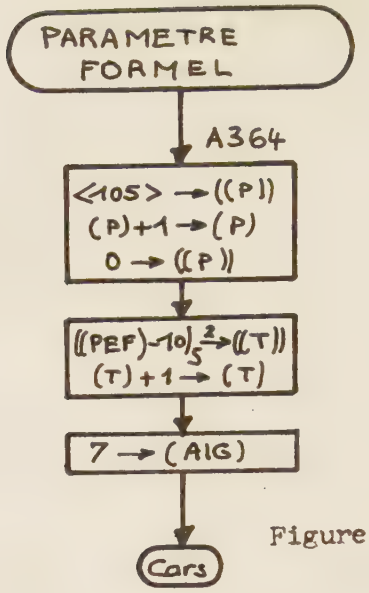


Figure VI-37

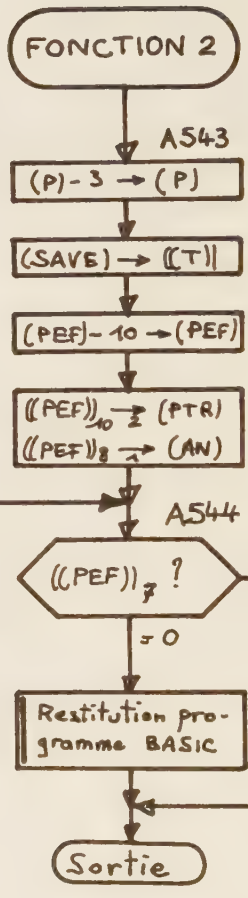


Figure VI-36

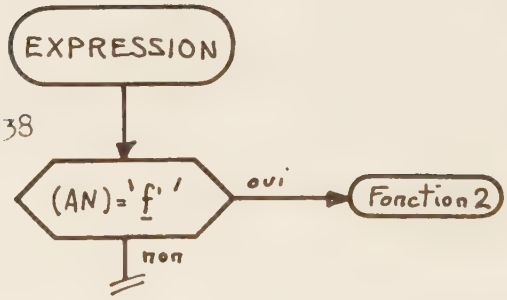


Figure VI-38

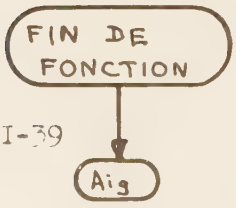


Figure VI-39

Si l'on reprend la figure VI-34, on voit que le code syntaxique <111> (qui symbolise une fonction prédéfinie) permet de séparer le traitement des fonctions prédéfinies des traitements des autres structures.

Pour effectuer ce traitement, on charge l'argument dans le registre accumulateur, on cherche le nom de la fonction prédéfinie, et on appelle le sous-programme correspondant, le résultat récupéré dans l'accumulateur est transmis en pile AUX.

Les fonctions internes sont traitées grâce aux 3 modules FØNCTIØN. Le premier module (figure VI-35), permet de stocker dans la cellule située à la 2ème position de la pile PEF, l'adresse de l'argument (qu'on prélève dans la pile T), le caractère sis en position d'analyse AN, et le contenu du pointeur caractère PTR. On descend la pile T d'un cran avant d'appeler la piste où l'on a enregistré les corps de toutes les fonctions définies. On cherche dans cette table, le corps de la fonction dont on a le nom dans ((PEF)-10), l'erreur 73 correspond au cas où l'on utilise une fonction qui n'a jamais été définie. Quand on a repéré la bonne fonction, on lance l'interprétation sur le corps de fonction en faisant pointer PTR sur le 1er caractère du corps de fonction (et en prenant quelques précautions sur les piles P et T).

L'interprétation du corps de fonction s'effectue comme pour une expression habituelle, on considère le paramètre formel comme un < primaire > (figure VI-37) et à chaque fois qu'on le rencontre, on transmet son adresse (qui est celle de l'argument) à la pile d'interprétation, en la prélevant dans la cellule de travail de la pile PEF. Le caractère spécial de fin de fonction (figure VI-39) joue le même rôle de délimiteur que le caractère de fin habituel, sauf qu'il permet au niveau de la routine "expression" (figure VI-38) de sortir de l'interprétation normale (la fonction interne calculée à l'aide de son argument est ramenée à une expression qui joue le rôle d'un primaire dans la suite de l'interprétation), et de revenir dans la routine FØNCTIØN au niveau du 2ème module.

Dans ce second module, on rétablit la pile P dans son état initial, on récupère le contenu de la cellule courante de la pile T (dont la position a varié au cours de l'interprétation de la fonction). La pile PEF est descendue d'un cran, et on restitue le caractère d'analyse et la valeur de PTR.

Le module FØNCTIØN 3, permet de savoir par la consultation de l'indicateur enregistré dans la cellule ((PEF)), si la pile est revenue à sa position initiale, auquel cas il faut restituer en MC le bloc courant du programme BASIC qui avait été remplacé par la pile. Ce module est commun aux deux types de fonctions.

VI-7-4 - Exemple

Reprenons les fonctions FND et FNC déjà vues :

```
10    DEF    FND(X) = P * X/180
20    DEF    FNC(X) = CØS (FND(X))
30    LET    A      = FNC (45)
```

VI-7-4-1 - Table des corps de fonctions

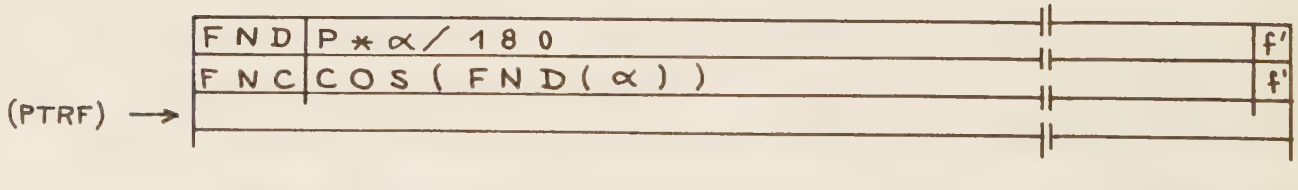


Figure VI-40

VI-7-4-2 - Evolution de la pile T, de la pile AUX et de la pile PEF

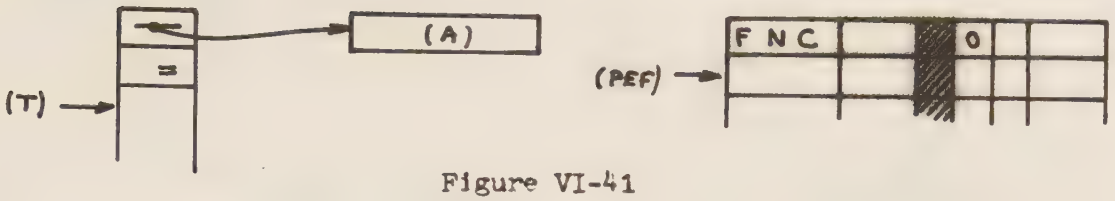


Figure VI-41

Situation après détection de la fonction FNG

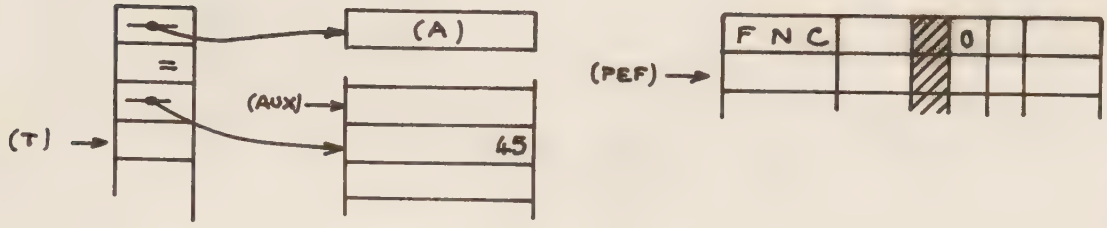


Figure VI-42

Situation à l'entrée dans le module FØNCTIØN 1

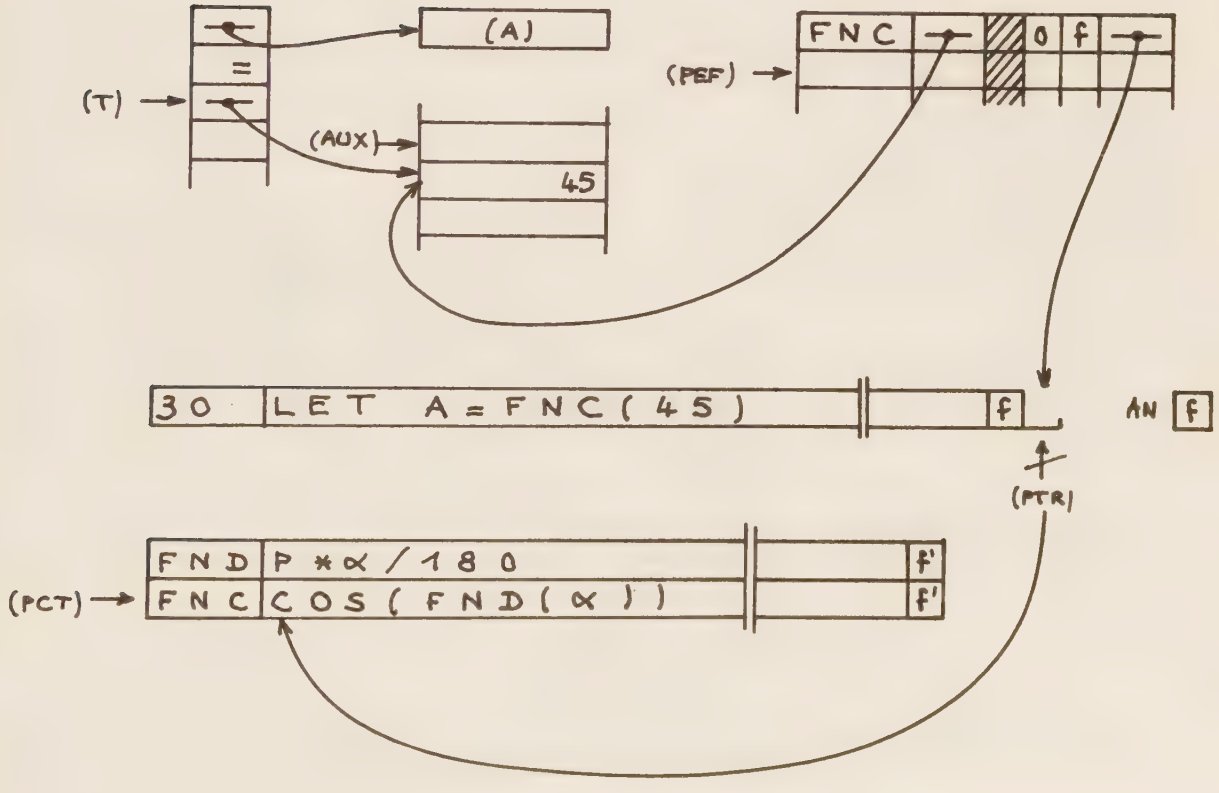


Figure VI-43

Situation à la sortie du module FØNCTIØN 1

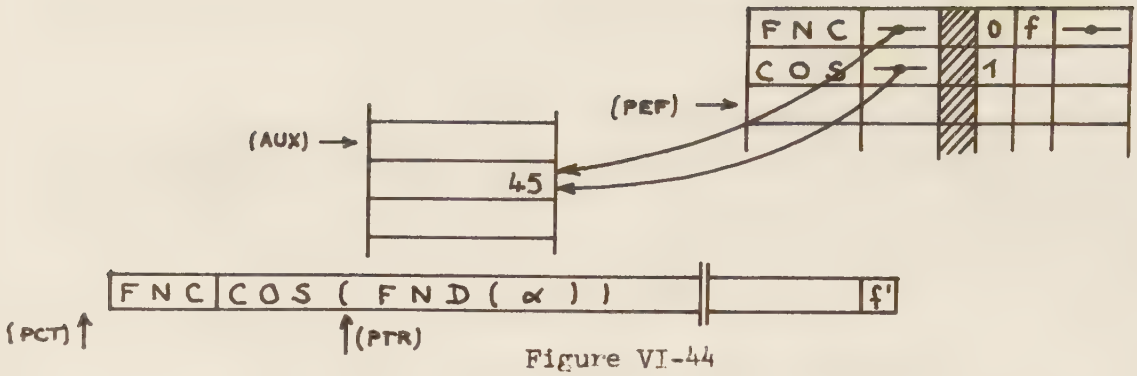


Figure VI-44

Situation après reconnaissance de CØS

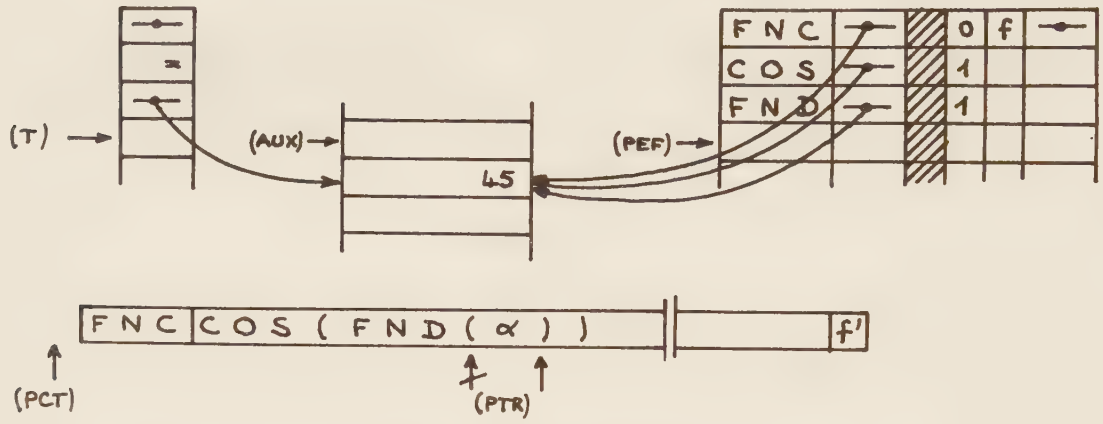


Figure VI-45

Situation après détection de END puis après 'α' à l'entrée dans FONCTION 1

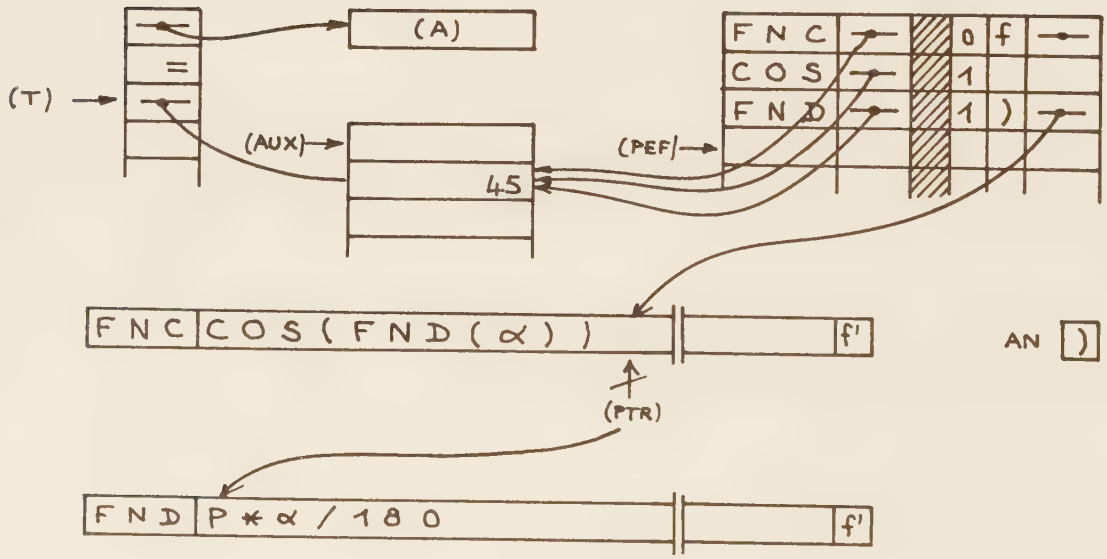


Figure VI-46

Situation à la sortie de FONCTION 1

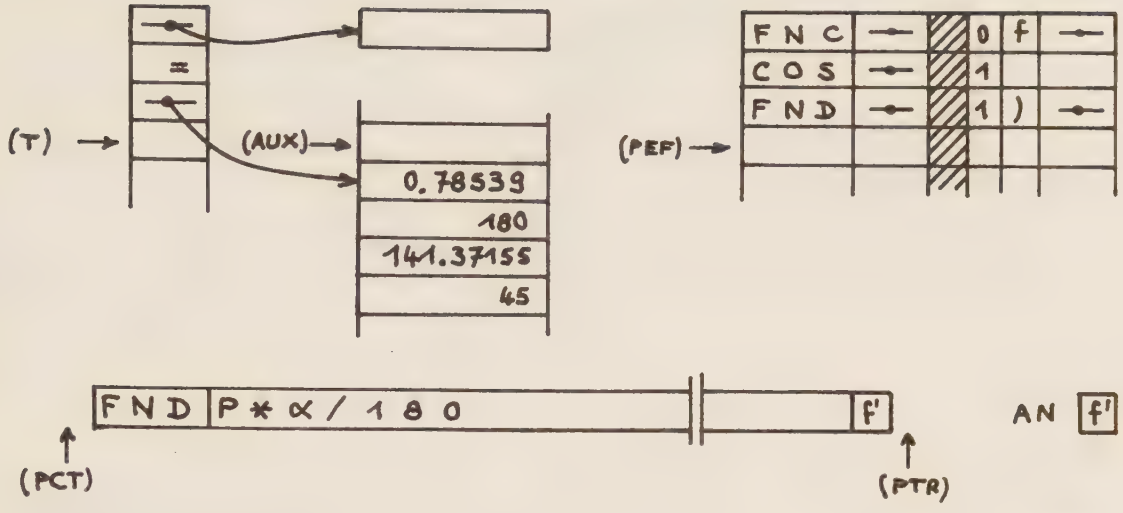


Figure VI-47

Situation à l'entrée dans FONCTION 2

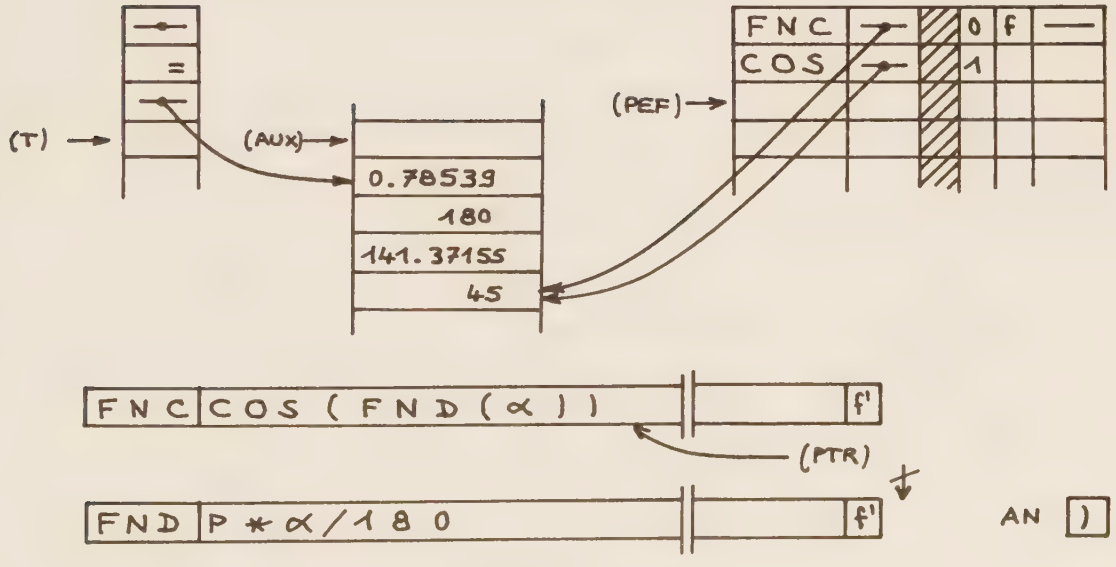


Figure VI-48

Situation après modules FONCTION 2 et 3

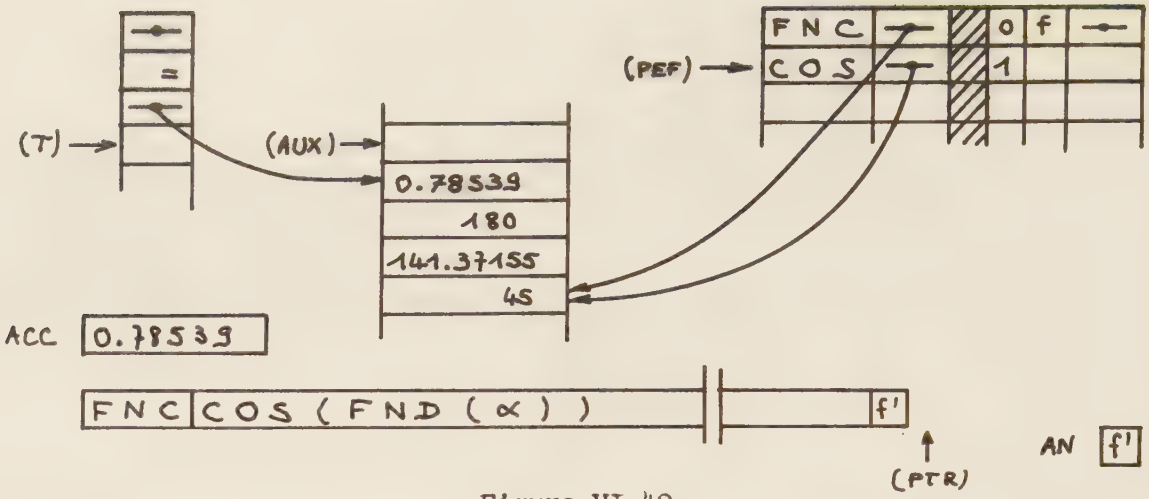


Figure VI-49

Situation à l'entrée dans S/P COS

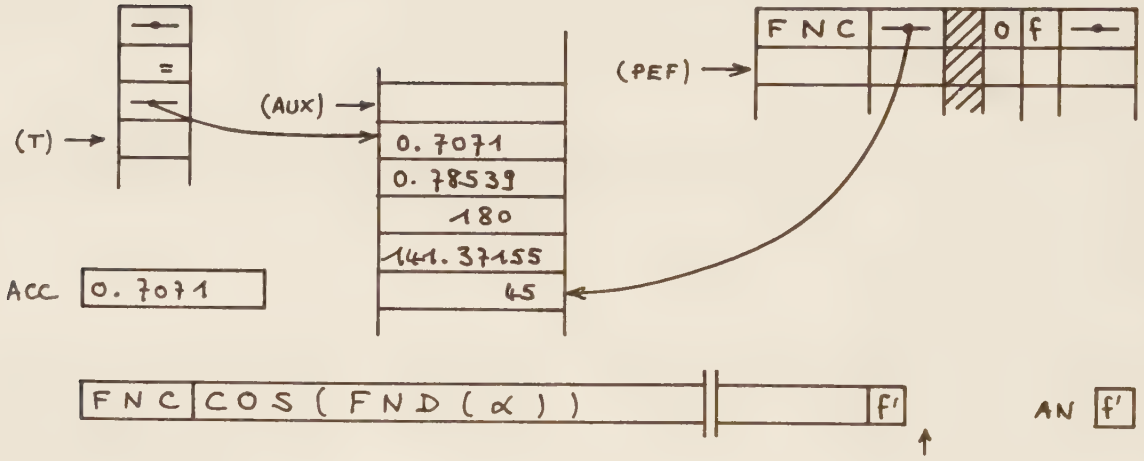


Figure VI-50

Situation après S/P COS et FONCTION 3 et à l'entrée de FONCTION 2

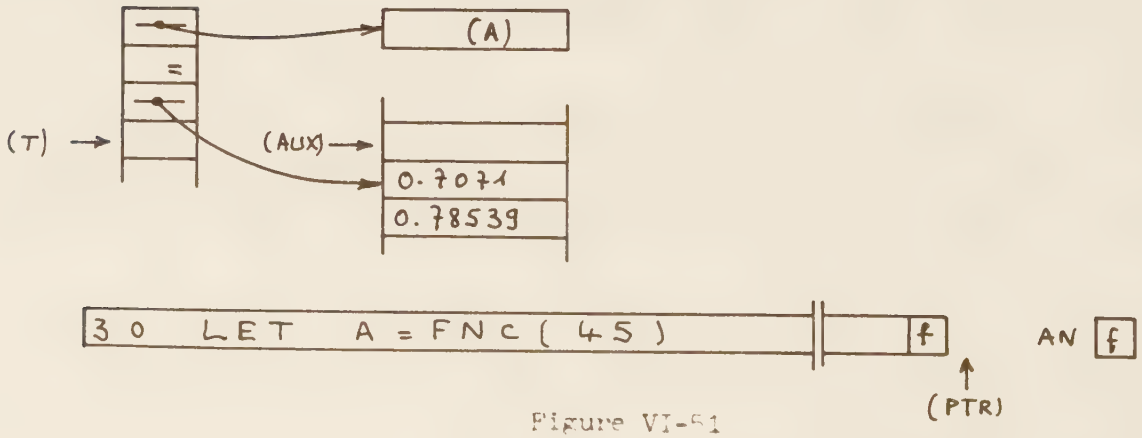


Figure VI-51

Situation après FONCTION 2 et FONCTION 3

VI-8 - OPERATIONS ARITHMETIQUES

L'ordre hiérarchique des symboles d'opération arithmétique s'établit comme suit :

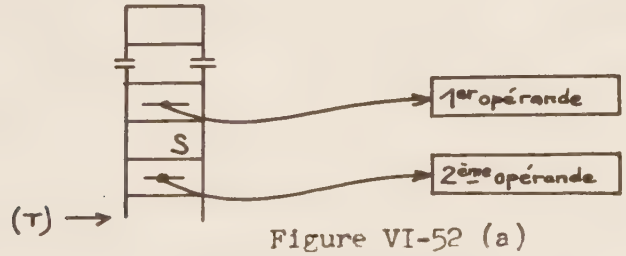
↑	symbole binaire
+ | -	symboles unaires
* | /	symboles binaires
+ | -	
MIN | MAX	

Le symbole unaire a une importance extrême, car selon la convention choisie, on obtient des évaluations numériques différentes de l'expression, par exemple, soit à calculer : - 2 ↑ 2

Les 2 solutions sont : $(-2)\uparrow 2 = 4$

$-(2\uparrow 2) = -4$

La solution retenue (comme l'indique l'ordre hiérarchique) est la seconde. Les modules d'interprétation effectuant les différentes opérations supposent les configurations suivantes de la pile d'interprétation.



S est le symbole d'opération

Figure VI-52 (a)

(a) : Opérations binaires

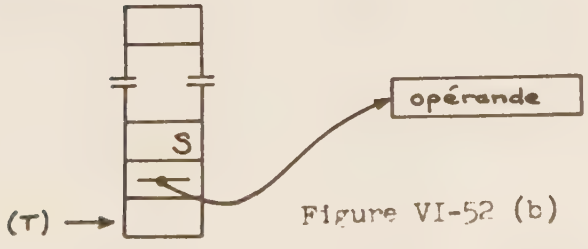


Figure VI-52 (b)

(b) : Opérations unaires

Le rangement en pile d'interprétation du symbole d'opération donne la possibilité d'empiler opérandes et symboles dans l'ordre où ils se présentent, les modules d'interprétation travaillant toujours avec les positions ((T)-1), ((T)-2) et ((T)-3).

VI-8-1 - Enregistrement des symboles en pile d'interprétation

Jusqu'à maintenant, on a examiné toutes les sortes d'opérandes utilisés en BASIC et on a vu à ce propos comment la pile d'interprétation recevait toujours les adresses ; l'enregistrement des symboles d'opération et le fonctionnement correspondant de la pile T se fait en deux temps.

Dans un premier temps, on enregistre le symbole dans la pile T dès qu'on a reconnu ce genre de caractères terminaux.

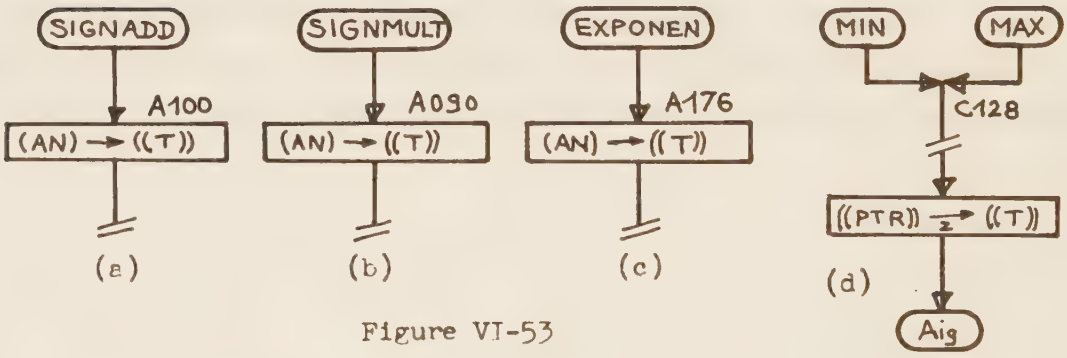


Figure VI-53

Le symbole d'opération reste dans la cellule du sommet de la pile tant qu'il joue par ailleurs pour l'interpréteur le rôle de délimiteur. Au moment où dans l'interprétation on reconnait une structure d'opérateur <opérande> < symbole d'opération> , on fait avancer la pile d'un cran puisqu'on ira chercher ensuite l'opérande suivant.

La reconnaissance des opérateurs s'obtenant par les réductions suivantes :

Dans "FACTEUR" <facteur> ↑ ⟶ <opérateur d'exponentiation>

Dans "TERME" <terme> < signe de multiplication> ⟶ <opérateur de multiplication>

Dans "SOMME S" < somme S > < signe d'addition > ⟶ <opérateur d'addition>

Dans "MINMAX" <minmax > < mineur-majeur > ⟶ <MinMax >

VI-8-2 - Insection des modules arithmétiques

Il faut insérer les modules aux endroits où l'on reconnait la structure : < opérateur > < opérande > .

Dans "PRIMAIRE" : <opérateur d'exponentiation > < primaire> ⟶ <facteur>
module MOD EXP

Dans "FACTEUR" : <signe addition > < facteur> ⟶ <terme>
module MØD CHS

<opérateur de multiplication > < facteur> ⟶ <terme>
module MØD MULT

Dans "TERME" : <opérateur d'addition > < terme > ⟶ <somme S >
module MØD ADD

Dans "SOMME" : < MinMax > < somme > ⟶ <minmax >
module MØD MINMAX

VI-8-3 - Modules arithmétiques

On sait que les modules arithmétiques trouvent dans la pile T, toutes les données nécessaires. Après le calcul, on envoie le résultat dans la pile AUX, dans une cellule dont l'adresse remplace en pile T celle du 1er opérande (ou le symbole en cas d'opération unaire).

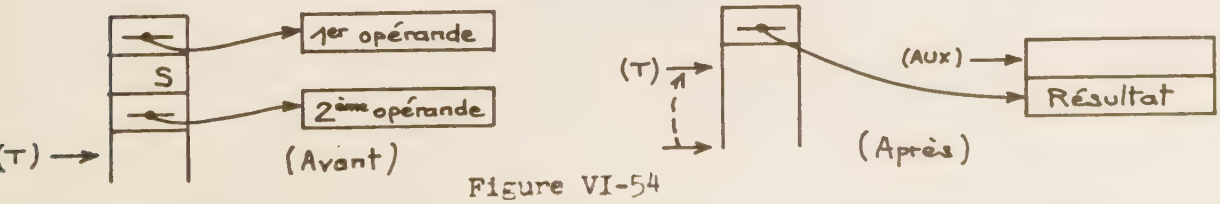


Figure VI-54

VI-8-4 - Organigrammes correspondants

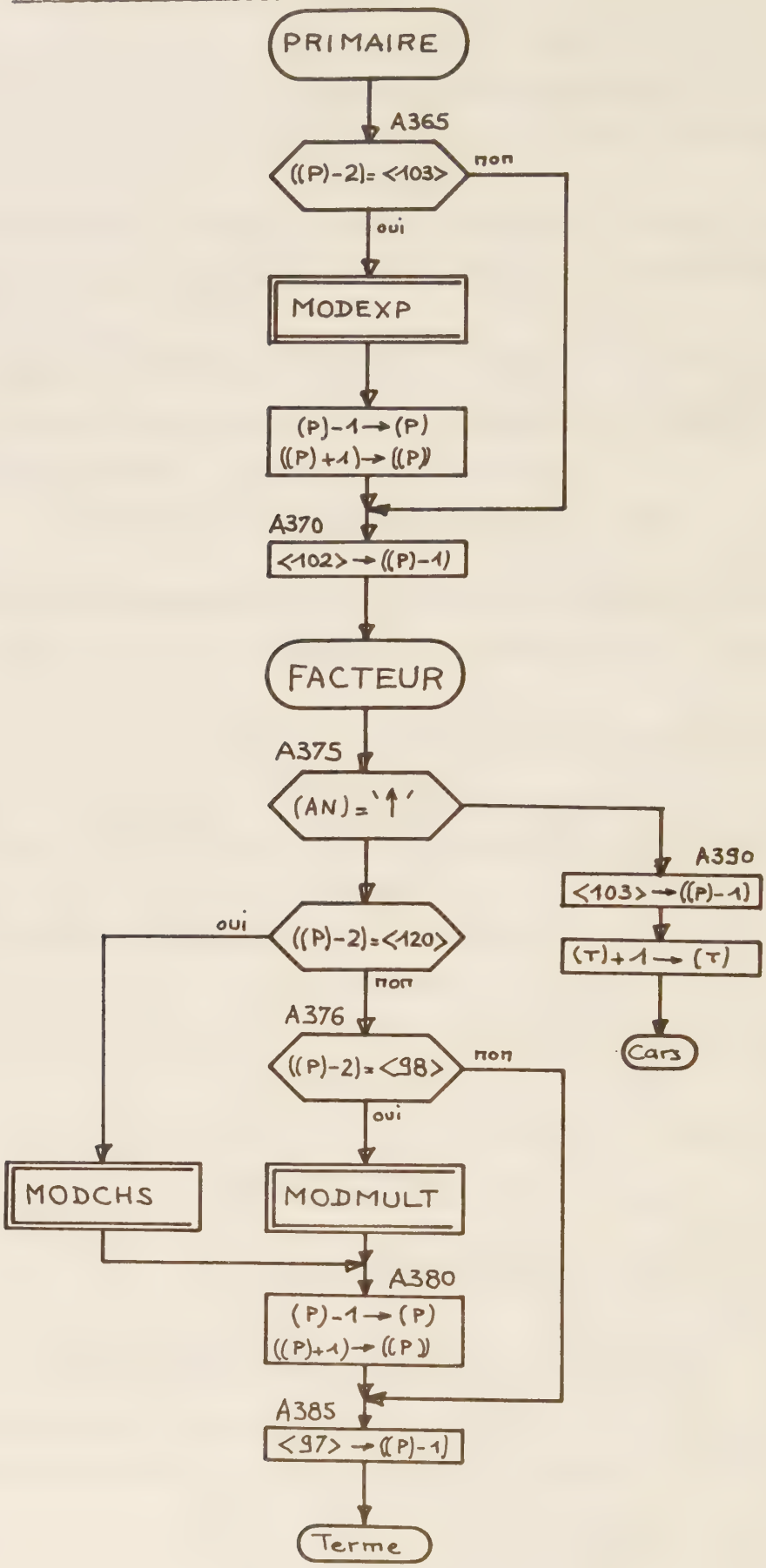


Figure VI-55

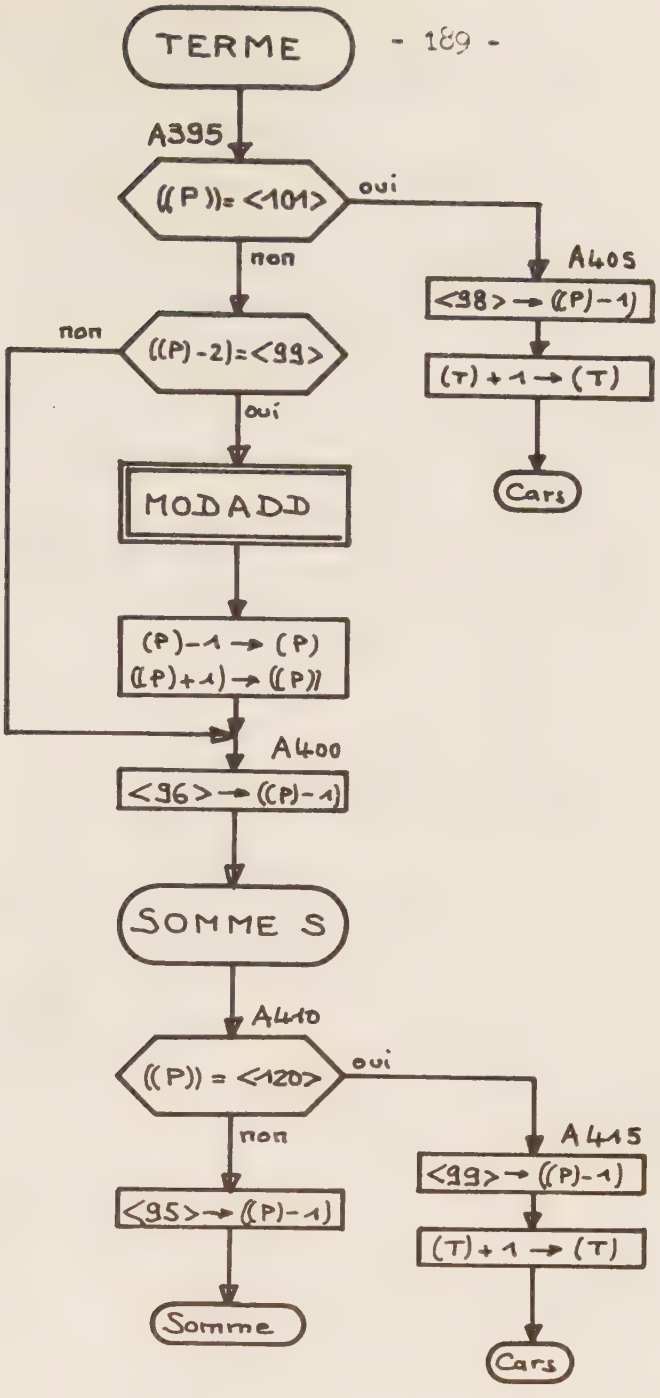

TERME
A395
((P))= <101>
oui
non
A405
<98> → ((P)-1)
(T)+1 → (T)
Cars
non
((P)-2)=<99>
oui
MODADD
(P)-1 → (P)
((P)+1) → ((P))
A400
<96> → ((P)-1)
SOMME S
A410
((P)) = <120>
oui
non
A415
<99> → ((P)-1)
(T)+1 → (T)
<95> → ((P)-1)
Somme
Cars

Figure VI-56

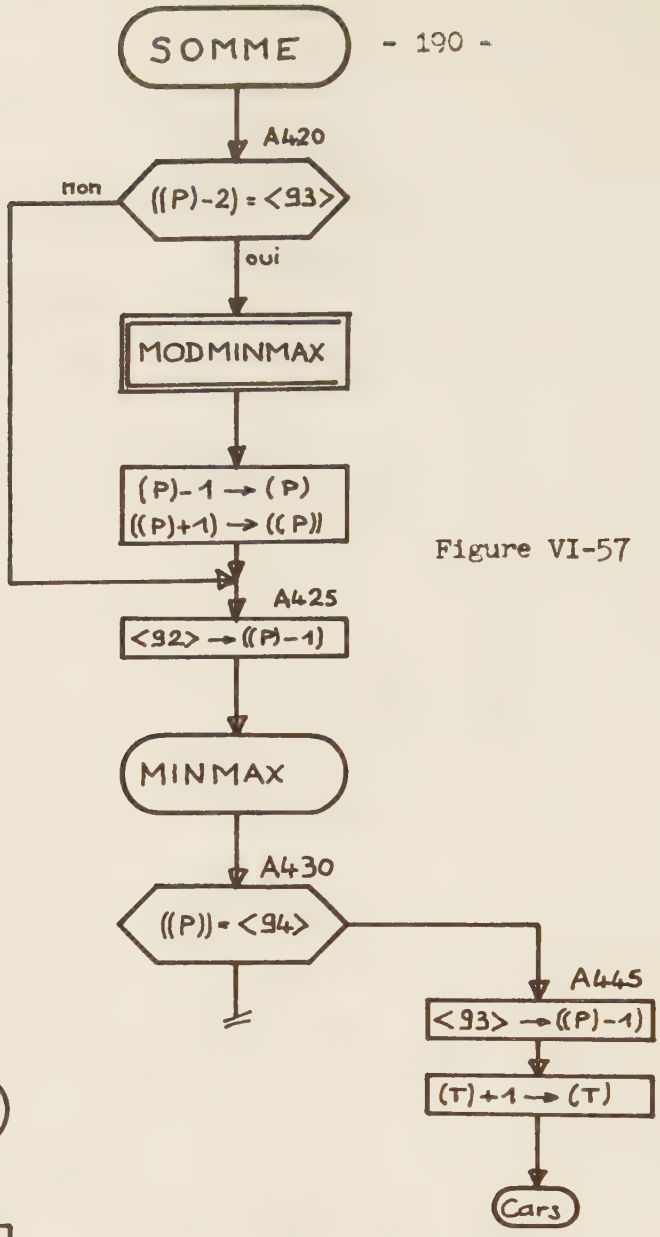


Figure VI-57

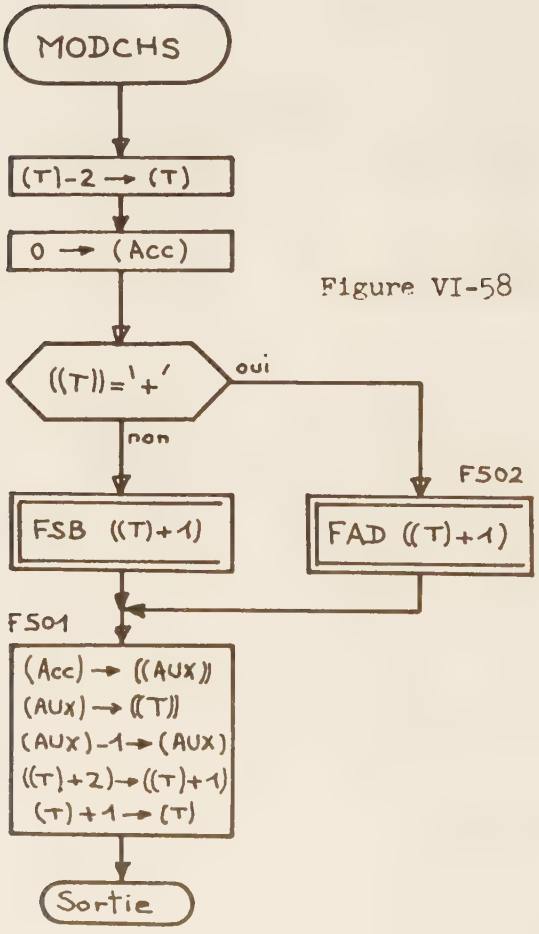


Figure VI-58

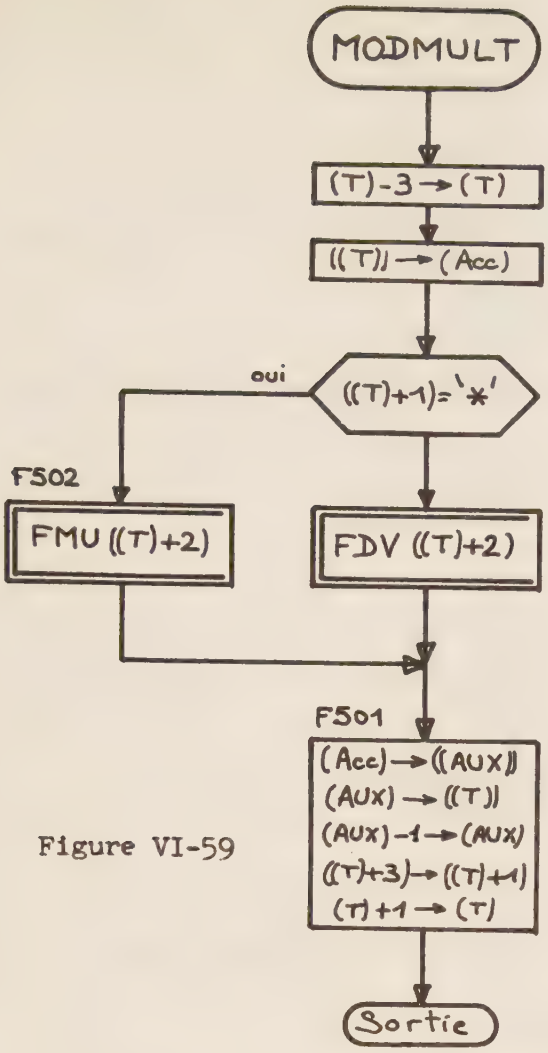


Figure VI-59

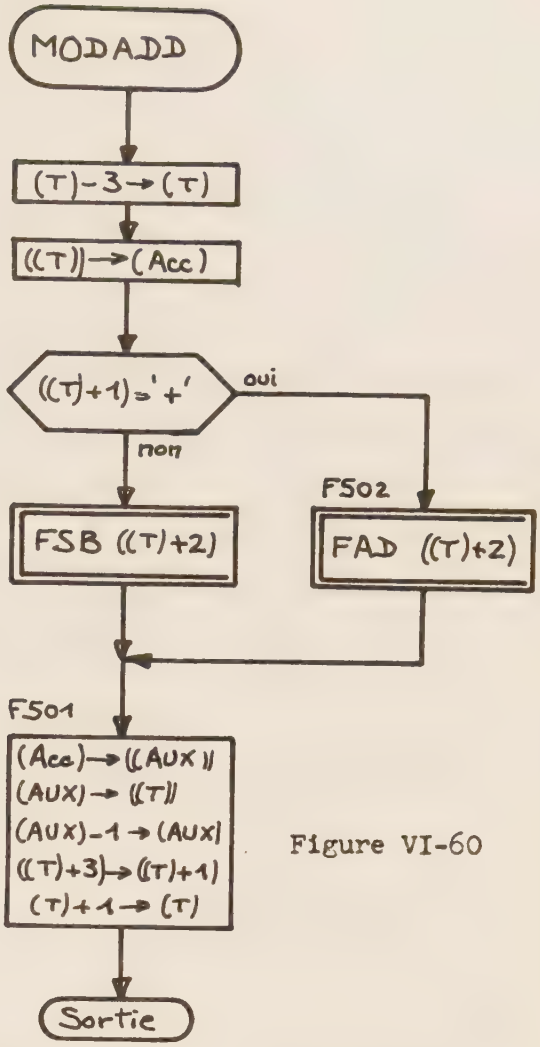


Figure VI-60

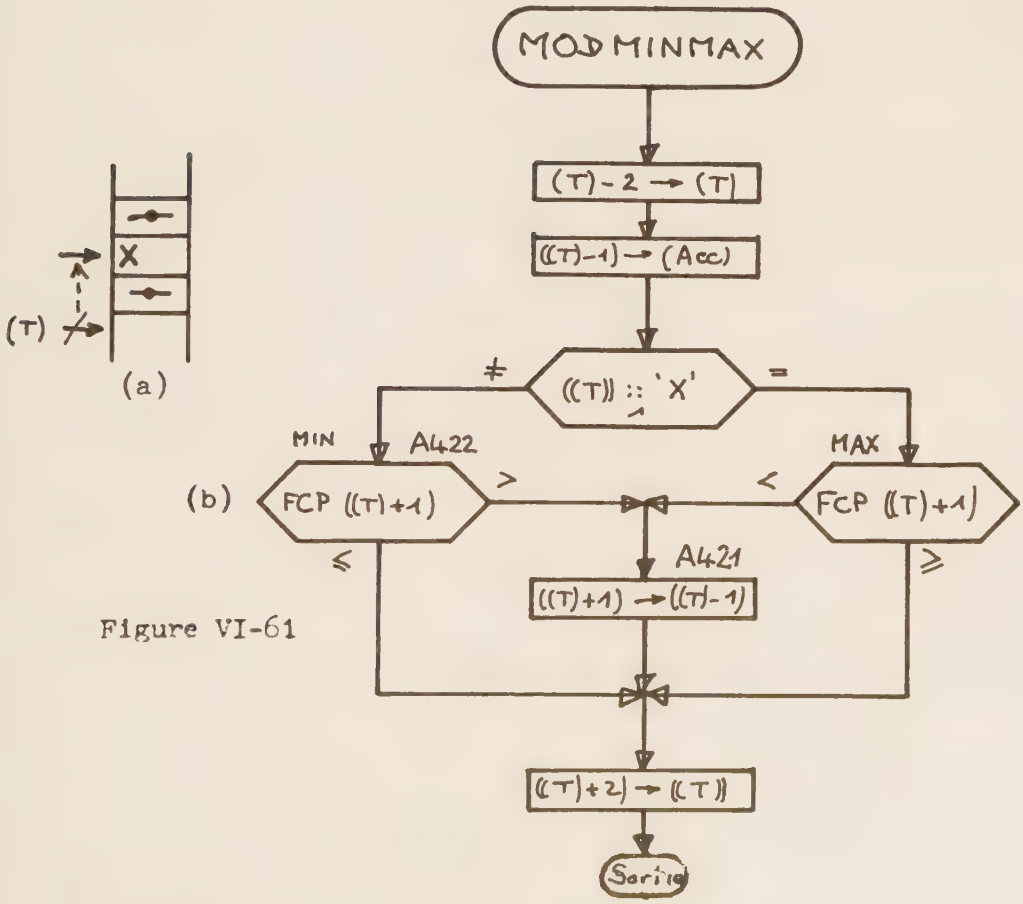


Figure VI-61

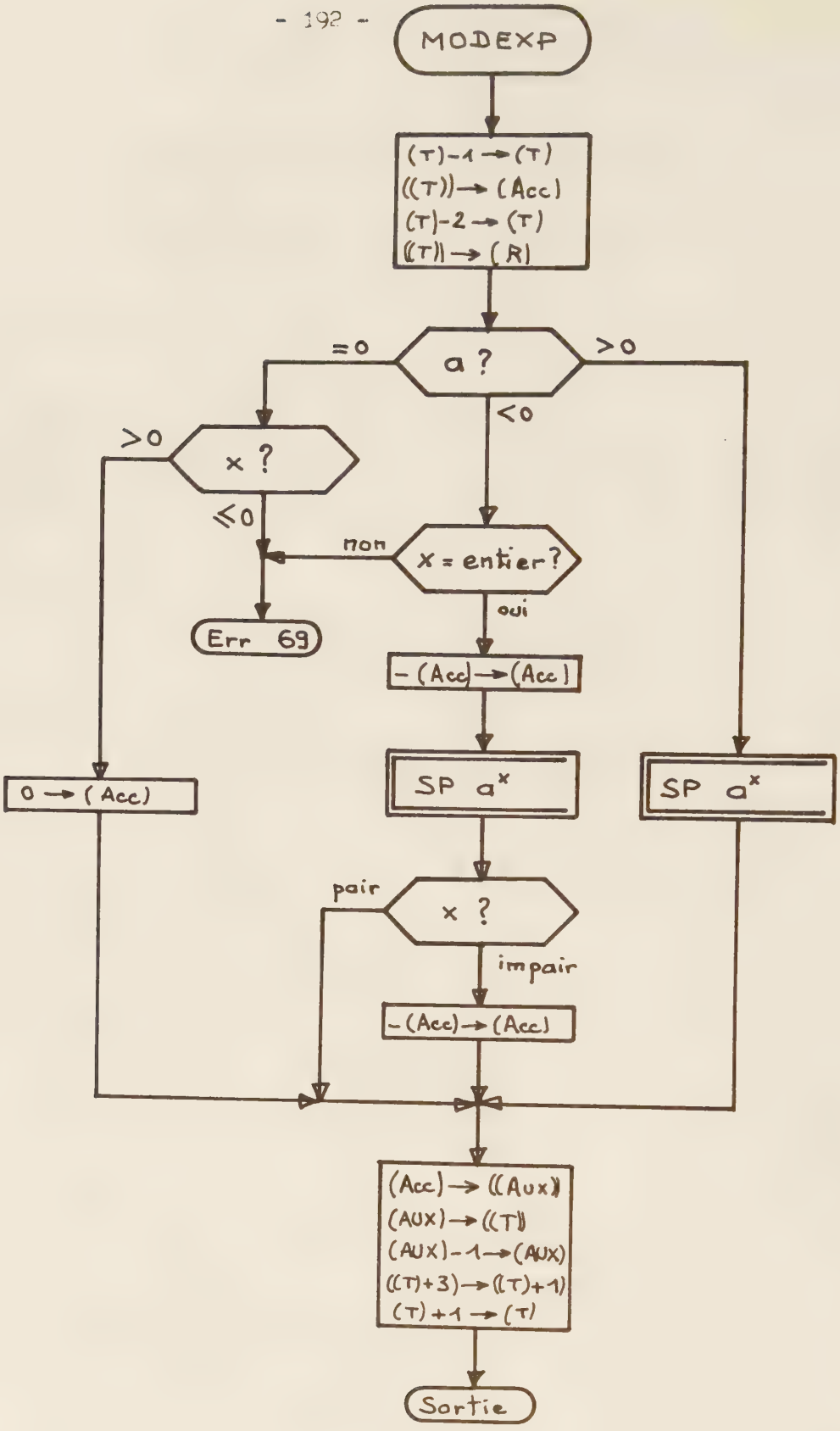

MODEXP
(T)-1 → (T)
((T)) → (Acc)
(T)-2 → (T)
((T)) → (R)
a ?
=0
>0
<0
x ?
>0
≤0
non
x = entier?
oui
Err 69
-(Acc) → (Acc)
SP a^x
SP a^x
0 → (Acc)
pair
x ?
impair
-(Acc) → (Acc)
(Acc) → ((Aux))
(AUX) → ((T))
(AUX)-1 → (AUX)
((T)+3) → ((T)+1)
(T)+1 → (T)
Sortie

Figure VI-62

Les noms de module FAD, FSB, FCP, FMU, FDV sont en fait les noms des macro-instructions d'arithmétique flottante.

Le traitement de l'exponentiation revient à calculer une quantité de la forme a^x, ce calcul est réalisé à l'aide du sous-programme a^x. Nous avons néanmoins apporté un certain nombre d'améliorations aux possibilités offertes par ce sous-programme.

Tel quel, le sous-programme fourni par le constructeur n'autorise pas des valeurs négatives pour a, ce qui est fâcheux, du moins quand l'exposant est entier.

Une étude de la fonction a^x dans l'ensemble des nombres réels nous apporte quelques solutions.

Soit $y = a^x$

1 $\underline{a > 0}$

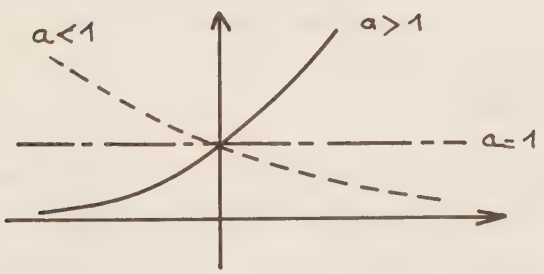


Figure VI-63

Il n'y a pas de problème, le sous-programme sait calculer.

2 $\underline{a = 0}$

a) $\underline{x > 0}$ $\quad x \text{ Log } a \longrightarrow -\infty \Longrightarrow a^x \longrightarrow 0^+ \qquad a^x = 0$

b) $\underline{x = 0}$ $\quad a^x = 1$

c) $\underline{x < 0}$ $\quad x \text{ Log } a \longrightarrow +\infty \Longrightarrow a^x \longrightarrow +\infty$ Sortie erreur

3 $a < 0$

a) x entier

on calcule $(-a)^n$

n pair	n impair
$a^n = (-a)^n$	$a^n = -(-a)^n$

b) x non entier

Cela revient à calculer le logarithme d'un nombre négatif, il n'y a pas de solution dans l'ensemble des réels.

D'où sortie erreur.

VI-9 - PRISE EN CHARGE DU PROGRAMME BASIC

On sait que les instructions du programme BASIC sont rangées par ordre numérique en tambour, elles sont regroupées par blocs de 9 instructions, chaque bloc étant sur une piste.

On prend en charge chaque instruction dans une zone de lecture en commençant par celle qui porte le plus petit numéro et on applique l'algorithme de traitement. Le traitement peut s'achever de différentes façons : on peut avoir une sortie erreur qui est traitée elle-même à part ; on peut avoir fonctionnement du programme en séquence, chaque instruction remplaçant au traitement celle qui la précède dans l'ordre numérique ; enfin il peut y avoir une rupture de séquence déclenchée par un certain nombre d'instructions.

Le fonctionnement en séquence ne pose pas de problèmes, on cherche dans le bloc d'instructions en mémoire, si l'instruction suivante s'y trouve, auquel cas on la prend, sinon on va chercher en tambour le bloc d'instructions suivant.

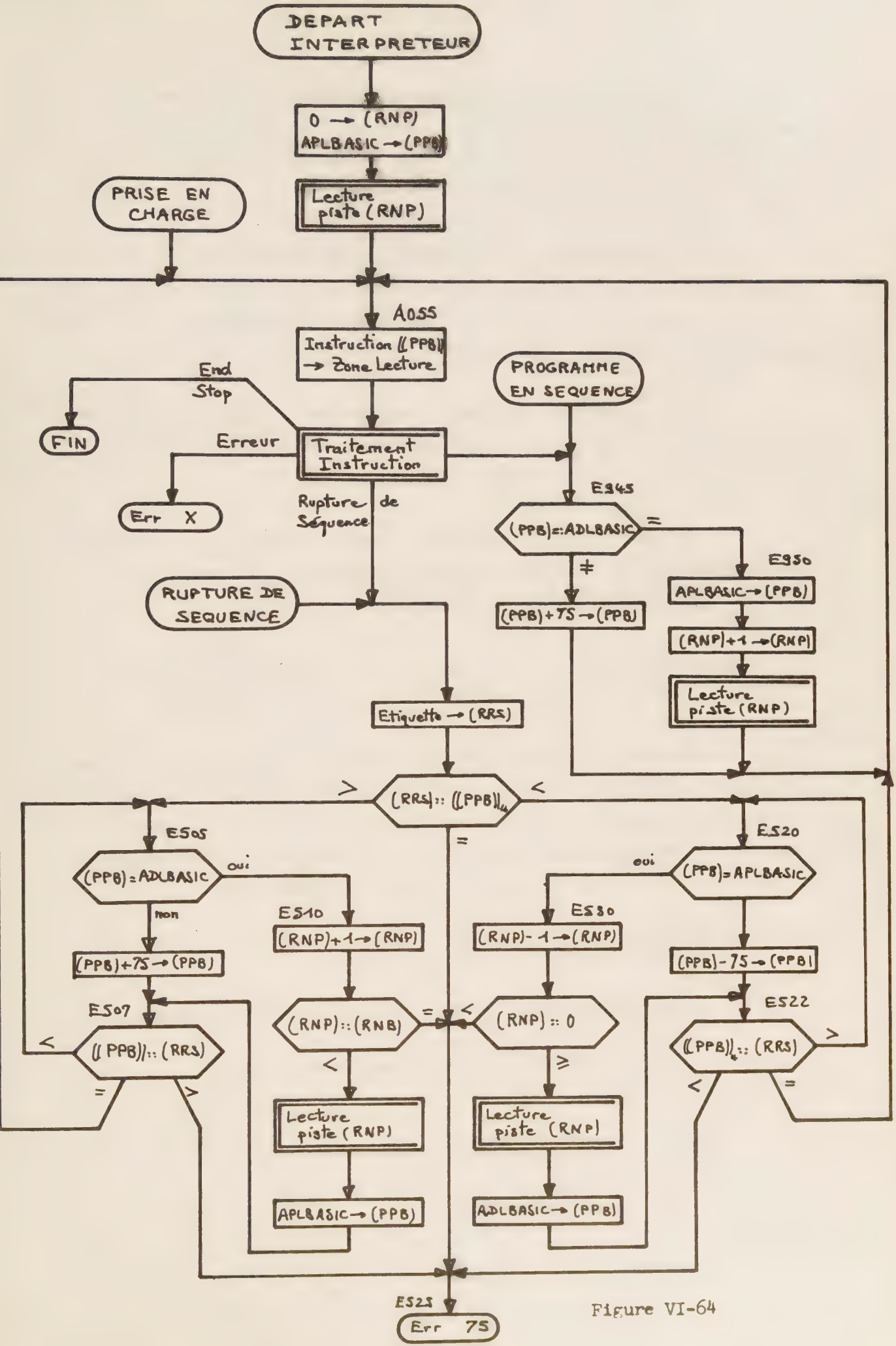

DEPART INTERPRETEUR
0 → (RNP)
APLBASIC → (PPB)
PRISE EN CHARGE
Lecture piste (RNP)
A055
Instruction ((PPB)) → Zone Lecture
End Stop
FIN
Erreur
Err X
Traitement Instruction
Rupture de Séquence
PROGRAMME EN SEQUENCE
E945
(PPB) = ADLBASIC
=
≠
E950
APLBASIC → (PPB)
(RNP)+1 → (RNP)
Lecture piste (RNP)
(PPB)+75 → (PPB)
RUPTURE DE SEQUENCE
Etiquette → (RRS)
(RRS) :: ((PPB))4
>
<
=
E505
(PPB) = ADLBASIC
oui
non
(PPB)+75 → (PPB)
E507
((PPB)) :: (RRS)
E510
(RNP)+1 → (RNP)
(RNP) :: (RNB)
Lecture piste (RNP)
APLBASIC → (PPB)
E520
(PPB) = APLBASIC
E530
(RNP)-1 → (RNP)
(RNP) :: 0
≥
Lecture piste (RNP)
ADLBASIC → (PPB)
(PPB)-75 → (PPB)
E522
((PPB))4 :: (RRS)
E525
Err 75

Figure VI-64

Le fonctionnement en rupture de séquence est plus délicat, car l'étiquette de renvoi peut être relative à une instruction située en avant ou en arrière de l'instruction en cours d'exécution, ce qui suppose deux routines de recherche.

Une sortie erreur est prévue pour une étiquette de renvoi égale à celle de l'instruction courante, ce qui signifie qu'on est dans une boucle sans fin, et pour le cas où l'étiquette de renvoi ne fait pas partie des étiquettes du programme.

Le registre RNP contient en permanence le numéro de la piste qui contient le bloc d'instructions courant, son évolution permet d'aller chercher un bloc désiré.

APLBASIC est l'adresse de la première ligne du bloc d'instructions quand on l'a amené en mémoire centrale, l'adresse de la dernière instruction du bloc est ADLBASIC.

PPB est un pointeur qui évolue à l'intérieur du bloc, en contenant l'adresse des instructions.

RRS est le registre rupture de séquence, rempli par programme au moment du traitement des instructions dont le résultat se traduit par un débranchement de la séquence normale. Il contient en format caractère le numéro de la ligne où l'on veut aller, et avec lequel on compare le numéro de l'instruction pointée par PPB.

VI-10 - INSTRUCTIONS READ ET INPUT

La forme générale de ce type d'ordre se présente comme suit :

<ordre READ>::= <READ> < liste de variables> avec :

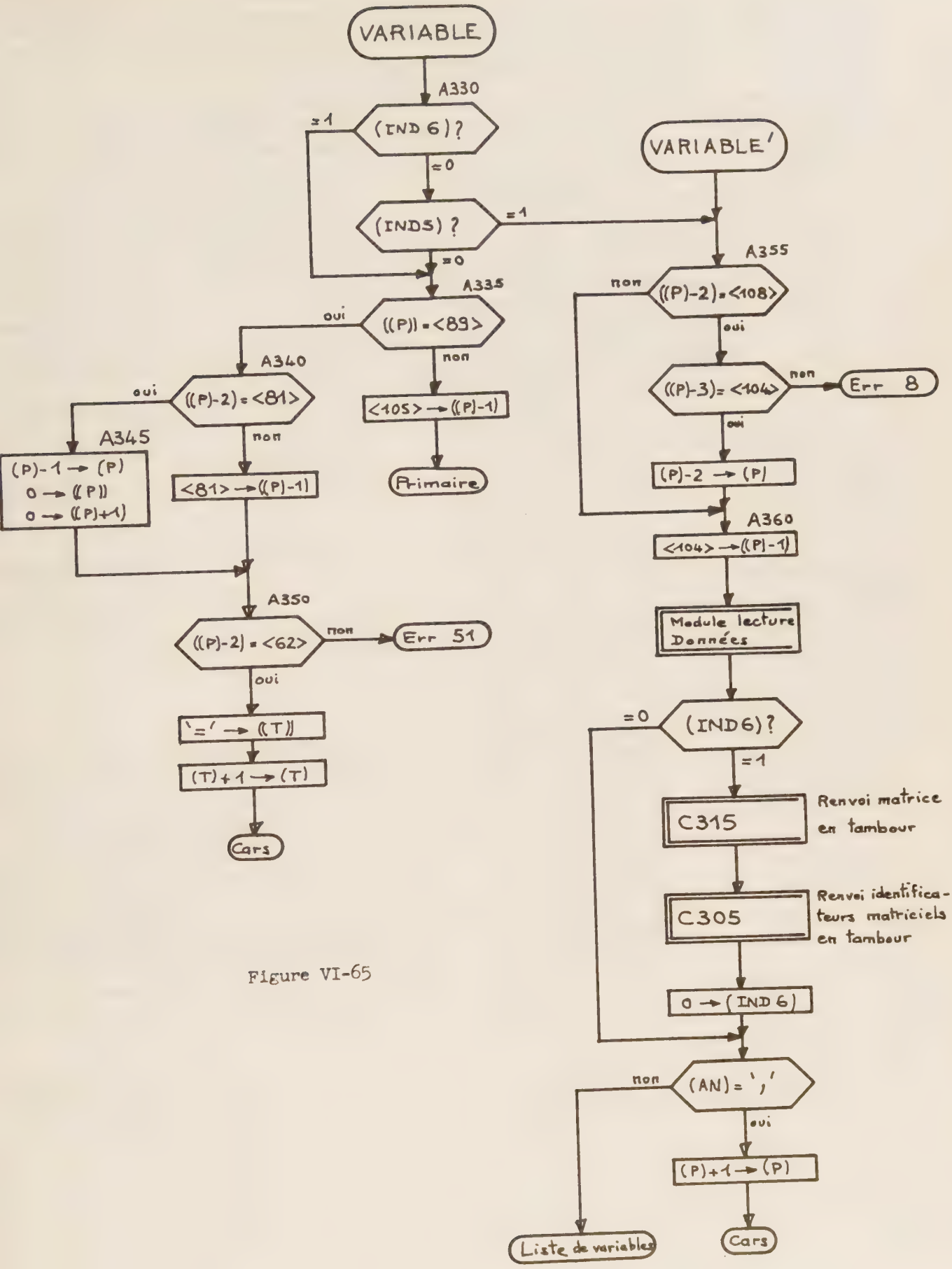

VARIABLE
A330
(IND 6) ?
=1
=0
(IND5) ?
=1
=0
A335
((P)) = <89>
oui
non
A340
((P)-2) = <81>
oui
non
A345
(P)-1 → (P)
0 → ((P))
0 → ((P)+1)
<81> → ((P)-1)
<105> → ((P)-1)
Primaire
A350
((P)-2) = <62>
non
Err 51
oui
'=' → ((T))
(T)+1 → (T)
Cars
VARIABLE'
A355
((P)-2) = <108>
non
oui
((P)-3) = <104>
non
Err 8
oui
(P)-2 → (P)
A360
<104> → ((P)-1)
Module lecture Données
=0
(IND6) ?
=1
C315
Renvoi matrice en tambour
C305
Renvoi identificateurs matriciels en tambour
0 → (IND 6)
non
(AN) = ','
oui
(P)+1 → (P)
Liste de variables
Cars

Figure VI-65

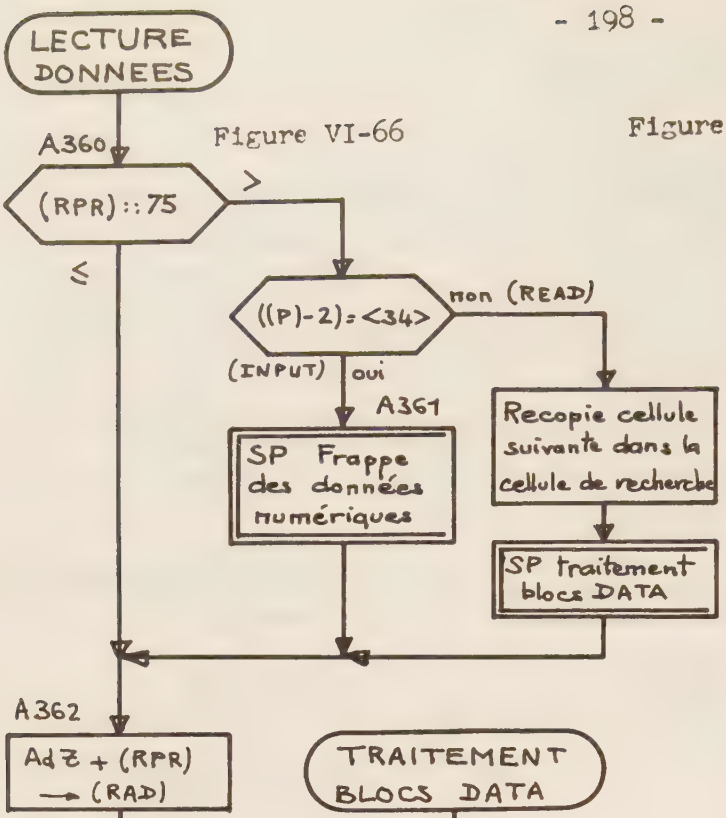


Figure VI-66

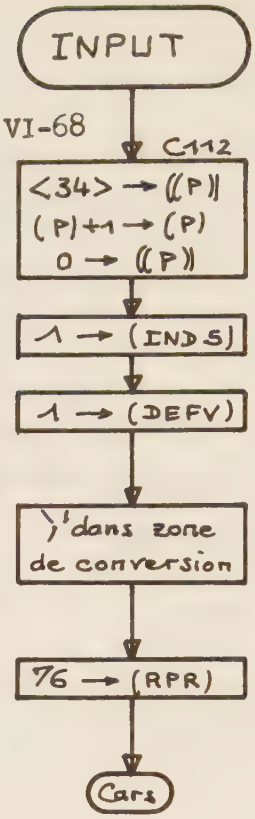


Figure VI-68

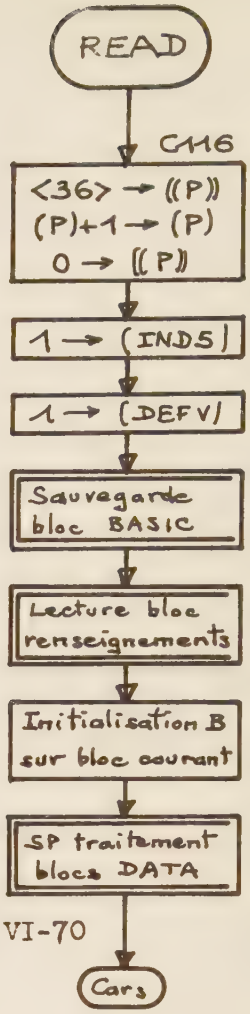


Figure VI-70

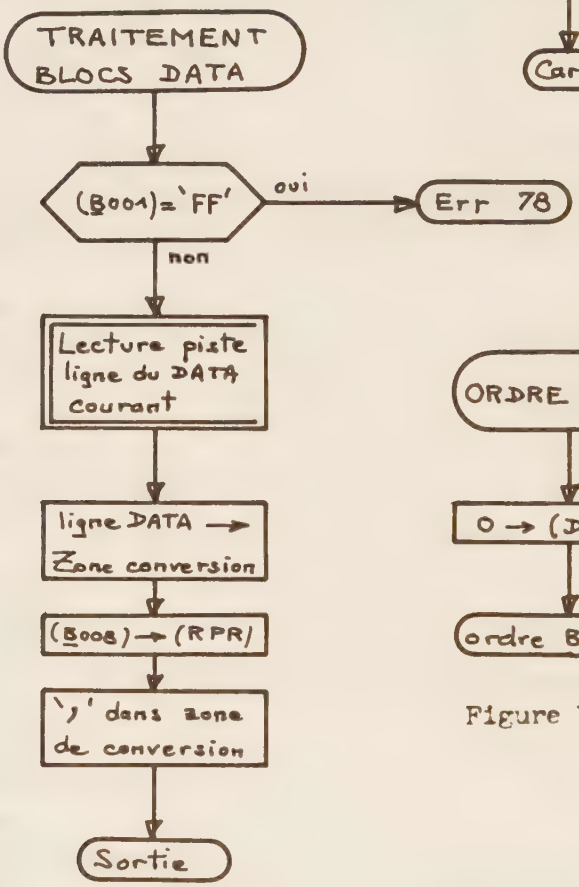


Figure VI-67

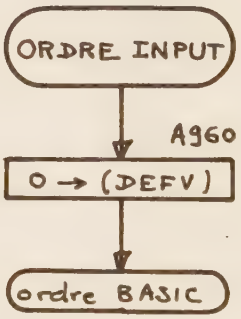


Figure VI-69

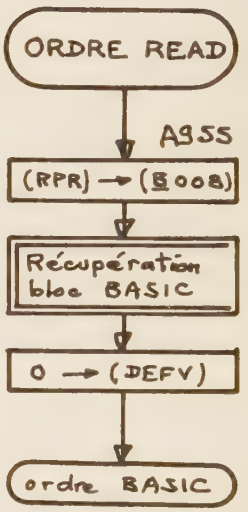


Figure VI-71

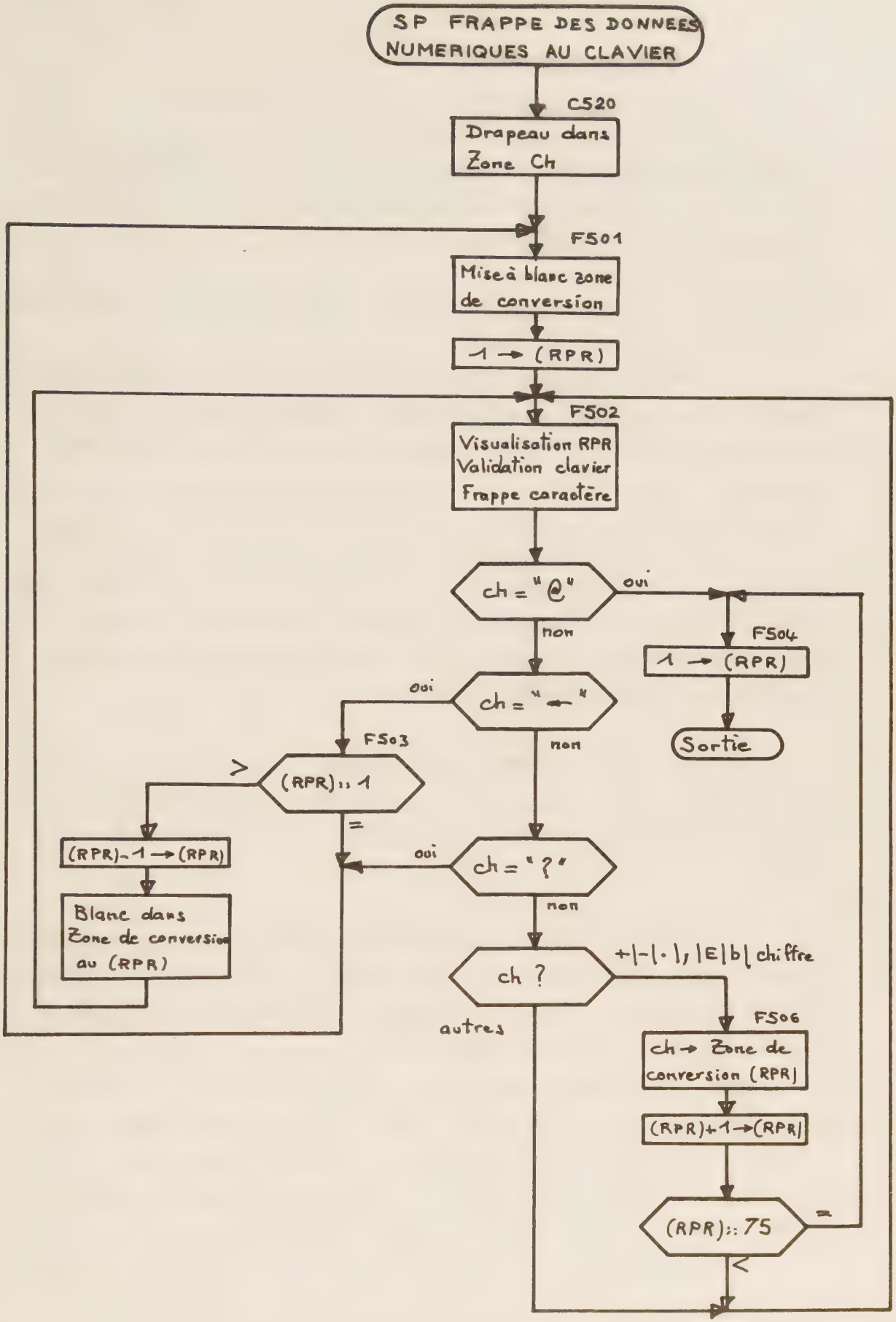

SP FRAPPE DES DONNEES NUMERIQUES AU CLAVIER
C520
Drapeau dans Zone Ch
F501
Mise à blanc zone de conversion
1 → (RPR)
F502
Visualisation RPR
Validation clavier
Frappe caractère
ch = "@"
oui
non
F504
1 → (RPR)
Sortie
ch = "←"
oui
non
F503
(RPR) :: 1
>
=
(RPR)-1 → (RPR)
Blanc dans Zone de conversion au (RPR)
ch = "?"
oui
non
ch ?
+|-|.|,|E|b|chiffre
autres
F506
ch → Zone de conversion (RPR)
(RPR)+1 → (RPR)
(RPR) :: 75
=
<

Figure VI-72

5 < liste de variables > < virgule > < variable > | < variable > ⟶
< liste de variables > où l'indicateur IND5 est positionné à 1 par les mots-clés READ et INPUT.

Le test de l'indicateur IND5 se fait au niveau de < variable > .

Le rôle d'un ordre READ est d'affecter aux variables figurant dans la liste, la valeur des constantes numériques figurant dans le DATA courant, et pour INPUT, la valeur des constantes frappées au clavier.

Au moment où l'on reconnait la structure d'une liste de variables, on peut commander soit la lecture dans le DATA courant dans le cas de READ, soit l'entrée de la quantité numérique au clavier alpha-numérique dans le cas de l'ordre INPUT. Ceci apparaît sous le nom unique de "module lecture données".

Si l'on fait abstraction un instant de IND6, la vérification de l'absence de la virgule dans AN, permet de clore la structure d'une liste de variables, sinon on va chercher dans l'instruction le caractère suivant.

Le rôle de l'indicateur IND6 est en fait d'éviter de considérer les variables apparaissant dans les indices d'une variable indicée, comme faisant partie de la liste des variables à affecter d'une part et à définir d'autre part (mise à 0 de DEFV).

L'indicateur est mis à 1 dans OUVPAR (figure VI-18) à la reconnaissance de la structure d'une variable indicée. Pour toute variable simple, on passe par VARIABLE (figure VI-65), IND6 étant à 1, on aboutit finalement à LISTE D'INDICES (figures VI-21 et VI-22), dans lequel on teste si IND5 est à 1, dans le cas positif la vérification que le compteur de parenthèses est à 0, permet de dire que la structure d'une variable indicée, sur laquelle on applique un ordre READ ou INPUT, est achevée. On peut alors remettre DEFV à 1, envoyer dans ((T)-1) l'adresse de l'élément à affecter et positionner AIG à 21, ce qui permet à la mise en charge de la virgule suivante (ou du caractère de fin) d'entrer dans VARIABLE′ (figure VI-65), d'affecter le résultat de la conversion de la constante numérique au bon élément et par IND6 égal à 1, de renvoyer la matrice en tambour ainsi que la table des identificateurs matriciels, enfin on remet IND6 à 0.

Dans LISTE D'INDICES en F507 (figure VI-22), si le compteur de parenthèses n'est pas égal à 0 (ce qui signifie que l'on est en présence de structures emboîtées), il faut reprendre le chemin normal F506.

Dans LECTURE DONNEES (figure VI-66), le test sur la valeur de RPR permet de savoir si le DATA courant est épuisé, auquel cas RPR contient 76, il faut donc dérouler le bloc de renseignements des DATA d'une chaîne et recopier la cellule suivante du bloc dans la cellule courante, en appliquant le module de traitement des blocs DATA sur le nouveau DATA. RPR est mis à 76 dans la routine INPUT (figure VI-68), ce qui oblige quand on reconnait que (RPR) > 75, à tester la pile d'analyse pour séparer les types d'ordres. Quand on reconnait < 34 > dans ((P)-2) on fait appel au sous-programme de frappe des données numériques qui présentera dans la zone de conversion une chaîne terminale de la même forme que celle qui est contenue dans un DATA. Dans tous les cas, on peut donc appliquer le sous-programme de conversion INTFLØ qui fournit dans l'accumulateur le résultat en flottant, résultat qu'il convient d'envoyer dans la cellule de 5 octets dont l'adresse est ((T)-1).

Pendant toute la durée de l'exécution de l'ordre READ, le bloc de renseignements sur les DATA est en MC. La portion de programme qui contient les lignes DATA prend en mémoire centrale, la place du bloc d'instructions BASIC. On prend d'abord la première cellule du bloc pour cellule courante (figure VI-70) et on applique le S/P de traitement des blocs DATA (figure VI-67) sur le DATA courant, ce qui consiste à vérifier que l'on n'a pas épuisé les DATA . Il faut alors aller chercher en tambour la piste qui contient la ligne DATA considérée, qu'on envoie en zone de conversion ; on prélève le pointeur de la position départ de conversion et on l'envoie dans RPR avant de clore la zone de conversion par une virgule.

Les routines ORDRE INPUT (figure VI-69) et ORDRE READ (figure VI-71) permettent de remettre DEFV à 0, et pour la dernière de mettre à jour la cellule de recherche DATA, puis de récupérer en mémoire centrale le bloc d'instructions BASIC.

Le sous-programme de frappe des données numériques au clavier a pour but de permettre l'entrée de quantités numériques en cours de programme. Il faut que les données ainsi introduites se présentent sous la même forme que les constantes apparaissant dans un DATA, en particulier, il faut vérifier que les quantités introduites sont constituées de caractères terminaux autorisés.

Si l'on examine l'organigramme (figure VI-72), on constate la création d'une zone Ch d'un octet limitée par un drapeau, dans laquelle vient se placer le caractère frappé au clavier, la chaîne numérique entre caractère par caractère (F506) dans la zone de conversion préalablement remise à blanc (F501). Le caractère ' @ ' est un caractère de commande, il signifie que l'introduction au clavier est terminée. D'autres caractères jouent un rôle de commande : ' ← ' symbolise le retour arrière d'un cran avec effacement du caractère erroné, '?' remet la zone de conversion à blanc.

VI-11 - INSTRUCTION RESTØRE

Le rôle d'un ordre RESTØRE est de permettre la réutilisation des données figurant dans les DATA.

Une instruction RESTØRE peut avoir un numéro de ligne cité après le mot-clé, c'est alors la cellule du bloc de renseignements qui comporte le même numéro qu'il faut recopier dans la cellule de recherche. S'il n'y a pas de numéro cité, on recopie la première cellule du bloc.

Le problème se pose de détecter les bornes de la chaîne constituant le numéro de ligne, pour cela, on repère la position du pointeur PTR au moment de la détection du mot-clé, et on la met en garde dans BØRNE1, on sait que dans ENTIER (figure VI-74), le fait que IND9 égale 1 indique la structure d'un numéro de ligne, la position de PTR est l'adresse du blanc qui suit le numéro.

Ex. : × × × × R E S T Ø R E 2 5 0

(BØRNE1) (BØRNE2)

L'utilisation de BØRNE1 et de BØRNE2 (qui reçoit la dernière adresse) permet de déclencher le changement de format (étendu en condensé) pour stocker sur 2 octets dans NØLINE le numéro de la ligne, exploitable dans la routine NØ LIGNE pour la recherche de la cellule convenable du bloc de renseignements.

Dans ENTIER, on vérifie la validité du numéro de ligne, c'est-à-dire sa composition d'un maximum de 4 chiffres. Dans NØ LIGNE (figure VI-75), en A715, on reconnait la structure d'un ordre RESTØRE, on appelle alors le bloc de renseignements DATA dans lequel on cherche la cellule comportant le même numéro que celui qui figure dans NØLINE, une fois la cellule trouvée, on la recopie dans la cellule de recherche et on renvoie le bloc. La détection de 'FF' signifie que le numéro de ligne

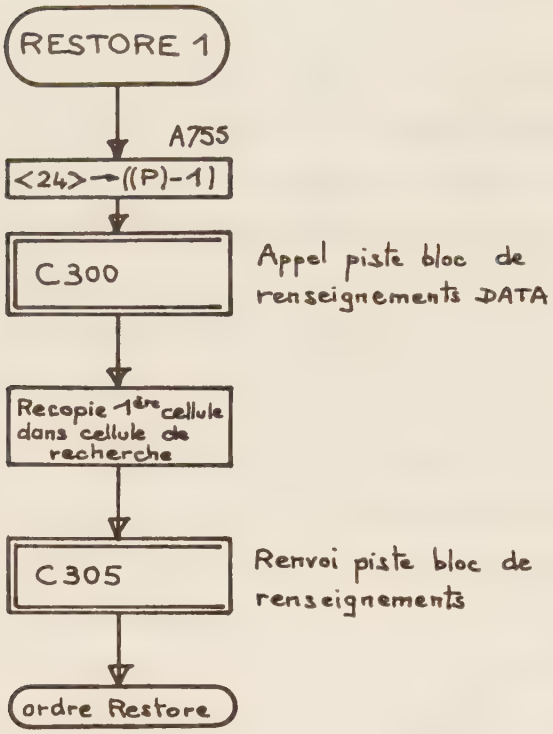


Figure VI-73

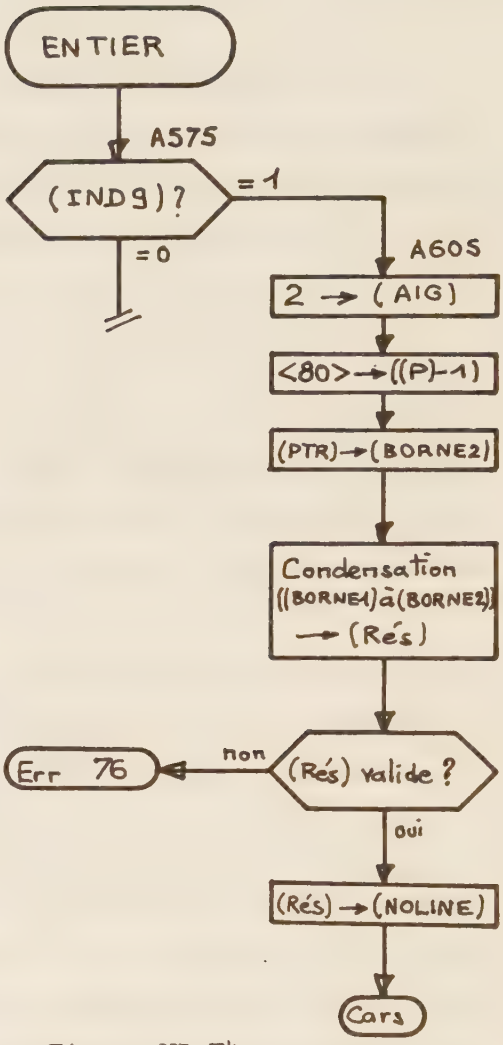


Figure VI-74

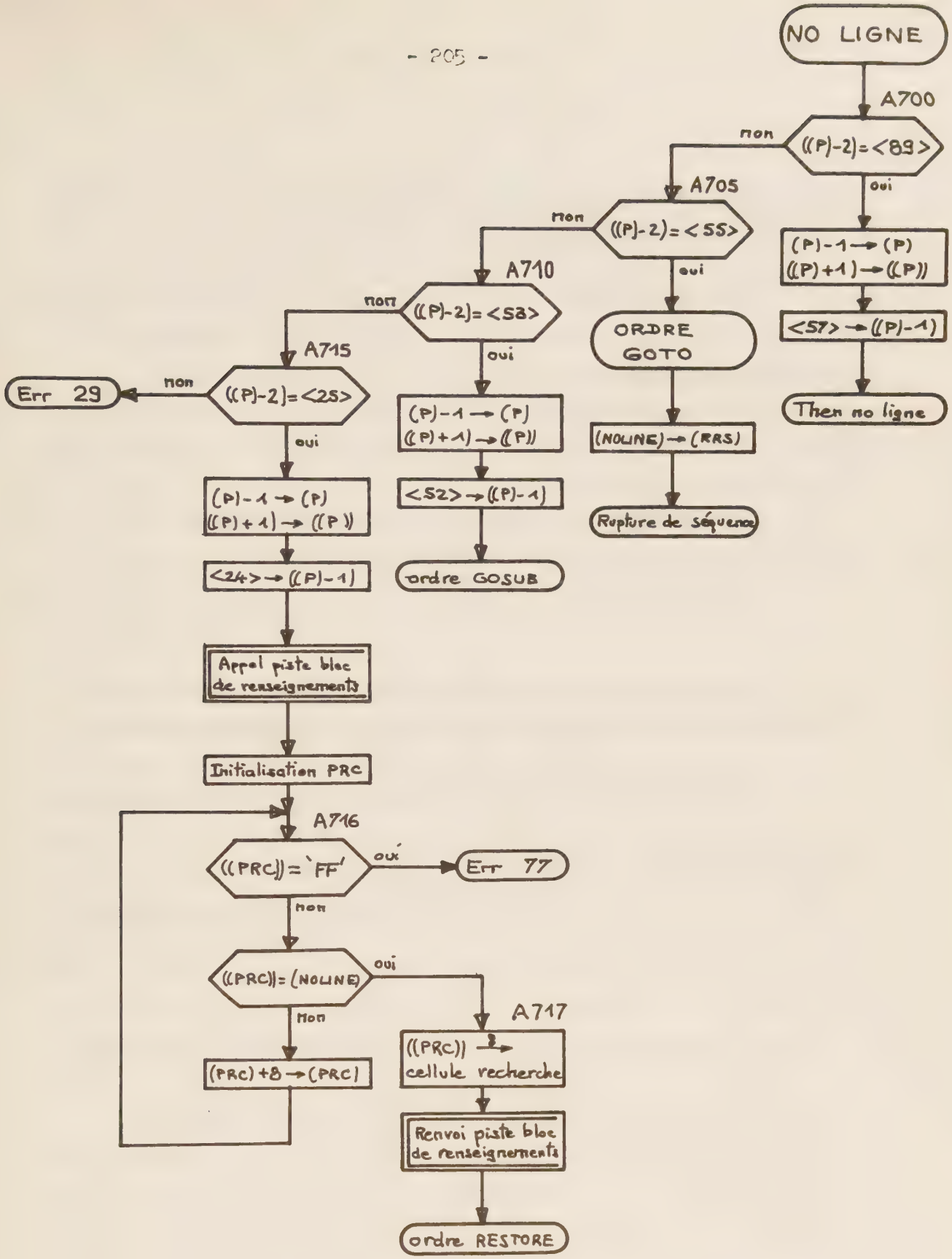

NO LIGNE
A700
((P)-2)=<89>
non
oui
A705
((P)-2)=<55>
non
oui
(P)-1→(P)
((P)+1)→((P))
<57>→((P)-1)
Then no ligne
A710
((P)-2)=<58>
non
oui
ORDRE GOTO
(NOLINE)→(RRS)
Rupture de séquence
A715
((P)-2)=<25>
non
Err 29
oui
(P)-1→(P)
((P)+1)→((P))
<52>→((P)-1)
ordre GOSUB
(P)-1→(P)
((P)+1)→((P))
<24>→((P)-1)
Appel piste bloc de renseignements
Initialisation PRC
A716
((PRC))='FF'
oui
Err 77
non
((PRC))=(NOLINE)
oui
non
(PRC)+8→(PRC)
A717
((PRC))→ cellule recherche
Renvoi piste bloc de renseignements
ordre RESTORE

Figure VI-75

cité dans l'instruction n'est pas valide puisqu'il ne correspond pas au numéro d'une ligne DATA.

RESTØRE 1 (figure VI-73) correspond au cas où il n'y a pas de numéro de ligne, on se contente alors de recopier le contenu de la première cellule du bloc.

VI-12 - ORDRE LET

La forme générale d'une instruction d'affectation est la suivante :

× × × × LET variable = variable = ... = variable = expression

Il y a donc la possibilité d'avoir une affectation multiple d'une même expression à différentes variables simples ou indicées.

L'exemple suivant permet de comprendre la méthode proposée :

Soit l'instruction : 150 LET A = B(I, J) = C = 5

on s'arrange pour que la pile d'interprétation ait la forme suivante :

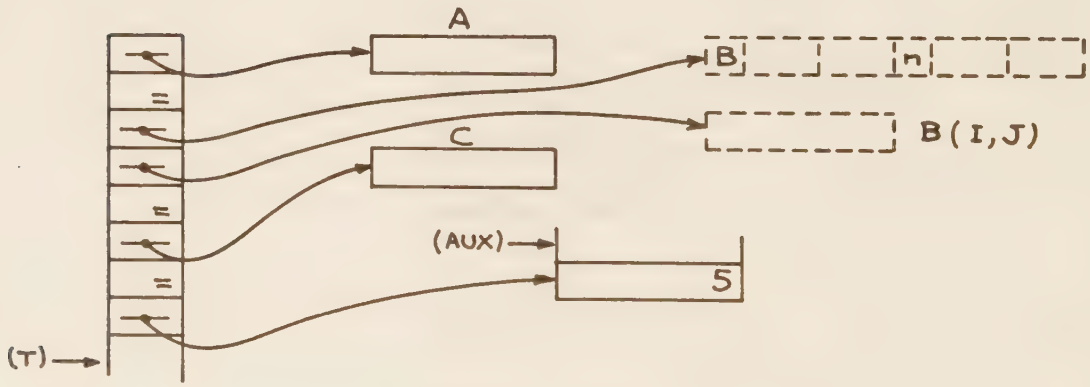


Figure VI-76

Il suffit maintenant de descendre la pile T, de façon à recopier dans les cellules pointées (successivement C, B(I,J) et A) le contenu de la cellule de la pile AUX qui contient la valeur de l'expression à affecter.

L'introduction dans la pile T, des adresses des variables simples ne pose pas de problème, c'est le principe même du traitement des variables simples tel qu'il a été exposé au chapitre VI-3. De même, l'adresse de la cellule qui contient la valeur de l'expression arithmétique est introduite en pile T, soit directement dans le cas des variables simples des quantités numériques, des éléments de tableaux, soit après traitement s'il y a des opérations arithmétiques à effectuer.

Le seul problème qui reste est celui de la mise en pile de l'adresse d'un élément de tableau auquel il faut affecter la valeur de l'expression. Ce problème trouve sa solution au niveau de la routine LISTE D'INDICES (figure VI-22), en effet IND8 positionné à 1 par le mot-clé LET, permet de passer par F508, la recherche du contexte postérieur permet de savoir alors si la variable indicée entre dans l'expression arithmétique (Ch est différent de "="), ou si on doit lui affecter la valeur de l'expression (Ch égale "="), on remet alors DEFV à 1, pour permettre l'affectation des variables simples qui peuvent suivre dans la partie gauche de l'instruction, et on introduit l'adresse de l'élément dans T, avant de faire progresser la pile.

La mise en place du symbole d'affectation en pile T se fait dans VARIABLE (figure VI-65). Remarquons que si l'on dispose dans T, de l'adresse de l'élément à affecter, par contre on ne dispose plus en mémoire centrale de la matrice qui contient l'élément. Il faut donc prévoir dans le module d'affectation (figure VI-77), la lecture de la piste des identificateurs matriciels. La consultation de l'identificateur

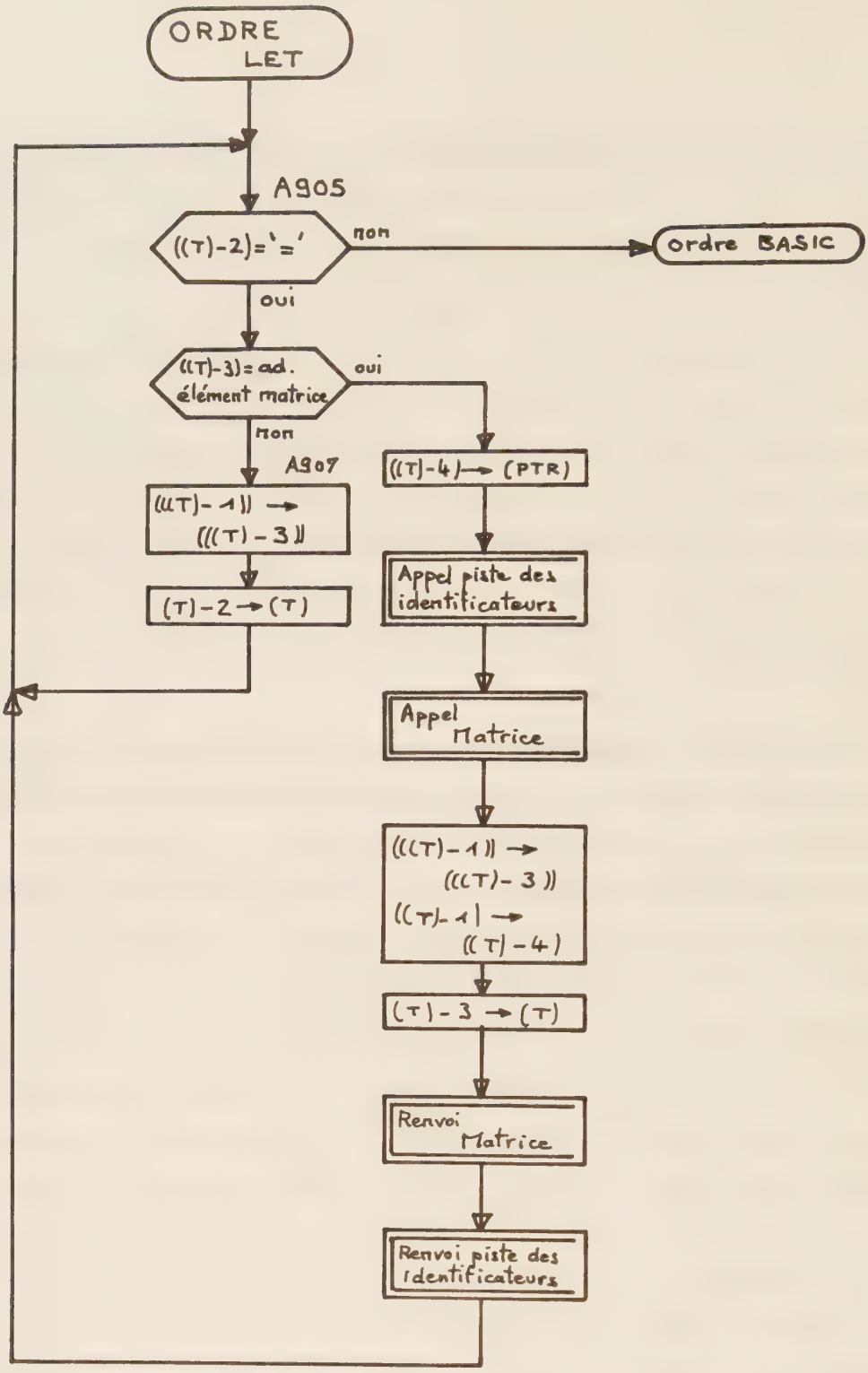

ORDRE LET
A905
((T)-2)='='
non
ordre BASIC
oui
((T)-3)= ad. élément matrice
oui
non
A907
((T)-4)→(PTR)
((T)-1)) → (((T)-3))
Appel piste des identificateurs
(T)-2→(T)
Appel Matrice
(((T)-1)) → (((T)-3))
((T)-1) → ((T)-4)
(T)-3→(T)
Renvoi Matrice
Renvoi piste des identificateurs

Figure VI-77

dont on a l'adresse, nous indique le numéro de la piste où se trouve la matrice qu'il faut lire. Après lecture on peut alors affecter correctement la valeur de l'expression à l'élément dont on a l'adresse en pile T.

La non-reconnaissance du symbole d'affectation dans ((T)-2) indique que nous avons fini les affectations et que nous pouvons aller prendre en charge l'instruction suivante (en passant par ordre BASIC).

Remarquons que de proche en proche, les cellules affectées reçoivent la valeur de l'expression, c'est pourquoi il n'y a pas recopie de l'adresse de la cellule AUX. Néanmoins, il faut recopier, dans le cas des variables indicées, l'adresse de la cellule affectée précédemment à la place de l'adresse de l'identificateur.

Nous terminerons avec l'ordre LET en reprenant la figure VI-6. La possibilité de l'affectation multiple oblige à tester la présence de l'élément syntaxique <62> , si cet élément est là, il faut maintenir l'indicateur de définition de variable simple DEFV à 1, tant qu'il y a un symbole d'affectation derrière.

VI-13 - ORDRE GØTØ

La forme générale d'un ordre GØTØ est :

× × × × GØTØ numéro de ligne

Cette instruction définit un branchement inconditionnel à l'instruction dont le numéro est cité.

Le mot-clé GØTØ en positionnant IND9 à 1, a permis à un <entier> de devenir <numéro de ligne> (figure VI-74). Dans la routine NØ LIGNE (figure VI-75), la détection du code <55> dans la pile P indique que l'on a affaire à un ordre GØTØ.

Il ne reste plus alors qu'à envoyer le numéro condensé figurant dans NØLINE, dans le registre RRS et à faire effectuer le branchement par la routine RUPTURE DE SEQUENCE (figure VI-64).

VI-14 - ORDRES FØR ET NEXT - ITERATIONS

VI-14-1 - Forme générale des ordres de boucle

× × × × FØR variable simple = expression initiale TØ expression finale

ou bien

× × × × FØR variable simple = expression initiale TØ expression finale STEP pas.

et

× × × × NEXT variable simple.

L'expression initiale, l'expression finale, le pas sont des expressions arithmétiques quelconques, positives ou négatives, entières ou non, la seule condition est que le pas doit avoir un signe tel que par incrémentation de la variable simple d'une quantité égale au pas, il y ait convergence vers la valeur de l'expression finale.

Remarquons que si le pas n'est pas expressément cité dans l'instruction, sa valeur implicite est 1.

La séquence d'instructions comprise entre les ordres FØR ... et NEXT d'une même boucle (la variable de boucle apparait dans les 2 ordres), est répétée autant de fois qu'il y a d'incrémentations (égales au pas) de la variable de boucle nécessaires pour que celle-ci atteigne la valeur de l'expression finale.

VI-14-2 - Identificateurs de boucle

Tant qu'on n'a pas épuisé une boucle, on a besoin en permanence de la valeur du pas et de la valeur de l'expression finale, ainsi que du numéro de la ligne qui suit l'ordre FØR, à laquelle il faut se brancher après exécution de l'ordre NEXT tant que la valeur finale n'est pas atteinte.

On peut mettre tous ces renseignements dans un identificateur de boucle qui sera empilé sur l'ordre FØR et désempilé sur l'ordre NEXT, après épuisement de la boucle.

Cet identificateur est composé comme suit :

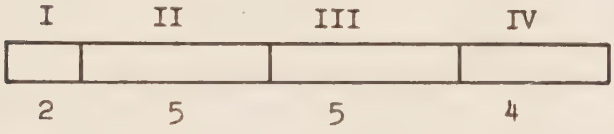


Figure VI-78

Le champ I reçoit le nom de la variable de boucle (2 caractères au maximum), le champ II la valeur du pas, le champ III la valeur finale, et le champ IV le numéro de la ligne où il faut boucler.

VI-14-3 - Traitement du FØR

Après la détection du symbole d'égalité, on dispose du nom de la variable de boucle dans les positions CALC1 et CALC2 que nous avons déjà utilisées, on peut donc créer dans la pile boucle la cellule qui contient le nouvel identificateur de boucle.

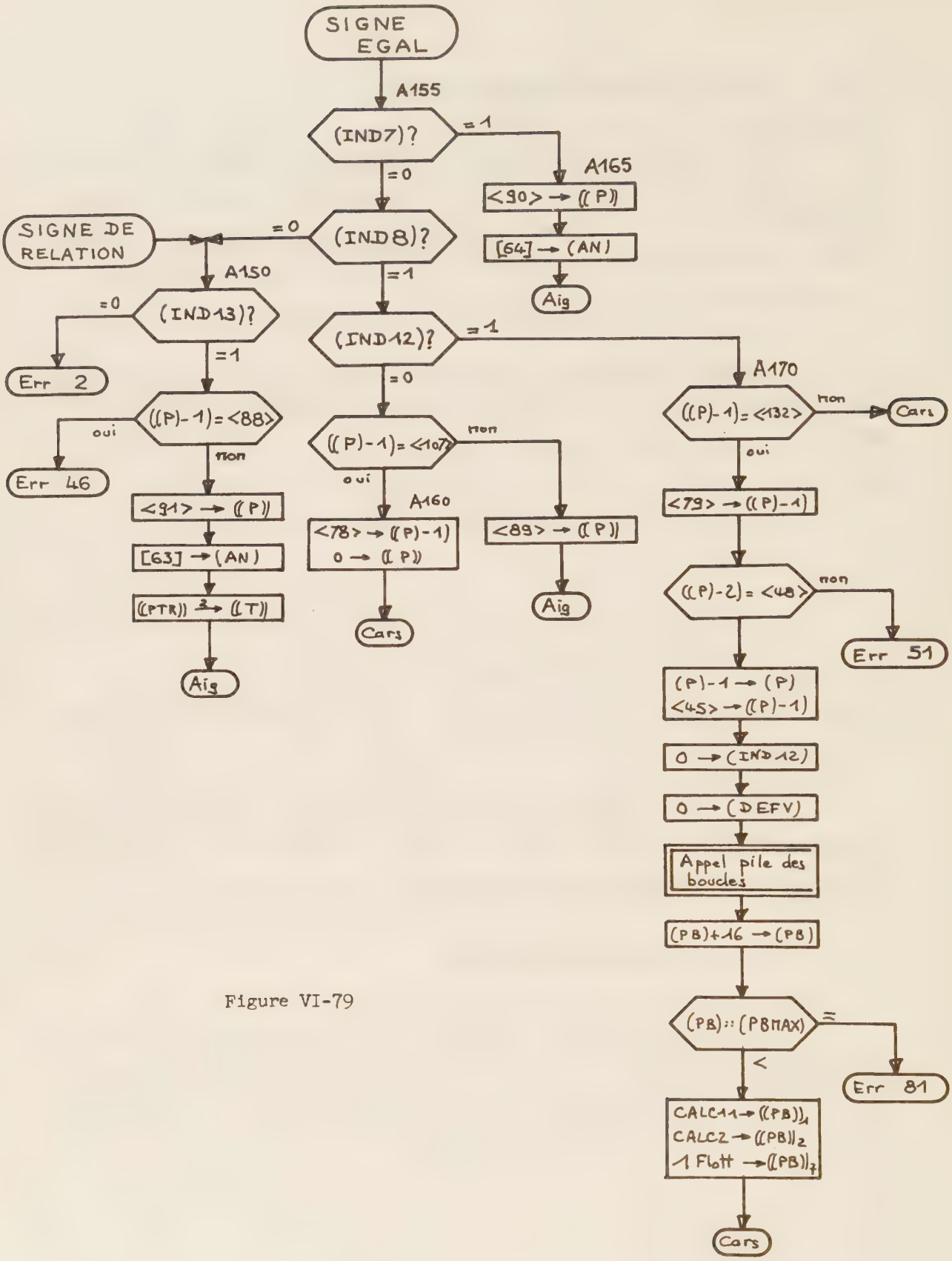

SIGNE EGAL
A155
(IND7)?
=1
=0
A165
<90> → ((P))
[64] → (AN)
Aig
SIGNE DE RELATION
=0
(IND8)?
=1
A150
(IND13)?
=0
=1
Err 2
((P)-1)=<88>
oui
non
Err 46
<91> → ((P))
[63] → (AN)
((PTR)) → ((T))
Aig
(IND12)?
=1
=0
((P)-1)=<107>
non
oui
A160
<78> → ((P)-1)
0 → ((P))
Cars
<89> → ((P))
Aig
A170
((P)-1)=<132>
non
Cars
oui
<79> → ((P)-1)
((P)-2)=<48>
non
Err 51
(P)-1 → (P)
<45> → ((P)-1)
0 → (IND12)
0 → (DEFV)
Appel pile des boucles
(PB)+16 → (PB)
(PB) :: (PBMAX)
=
<
Err 81
CALC11 → ((PB))1
CALC2 → ((PB))2
1 Flott → ((PB))3
Cars

Figure VI-79

On peut remarquer que dans la filière A170 (IND8 = IND12 = 1), on remet DEFV à 0, puisque la variable simple de boucle est la seule variable de l'instruction FØR qui peut être définie (ou redéfinie). Après l'appel de la piste qui contient la pile des boucles, on vérifie qu'il n'y a pas trop de boucles emboîtées. La mise en place du nom de la variable de boucle s'accompagne de la mise à 1 en flottant de la quantité qui représente le pas dans l'identificateur. S'il y a un pas cité dans l'instruction, sa valeur remplacera le 1 qu'on vient de mettre,de façon que le pas implicite de 1 soit déjà en place.

La pile T nous permet le traitement de tous les cas. Après la compilation de l'expression initiale, elle a l'aspect suivant :

Figure VI-80

Après initialisation de la variable de boucle, on tasse la pile vers le bas en supprimant l'adresse de la variable de boucle.

Figure VI-81

Avant d'entrer dans ORDRE FOR, la pile T a la forme suivante :

Figure VI-82

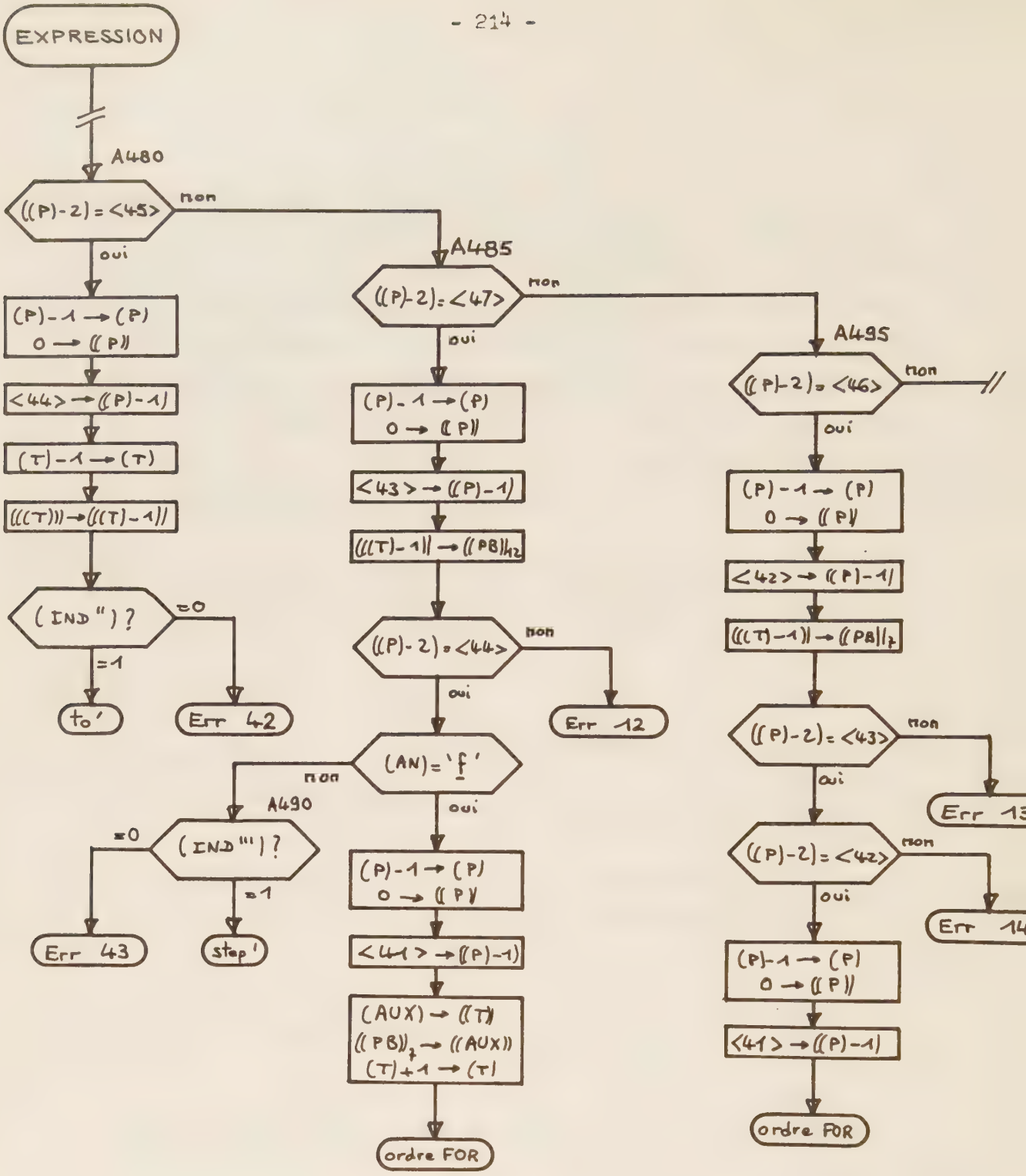


Figure VI-83

Traitement de valeur initiale, valeur finale et pas

Remarquons que selon la présence ou non de "STEP pas ", l'introduction de l'adresse du pas dans la pile, se présente de 2 façons. Si le pas est cité, sa compilation nécessite la présence de son adresse dans T, il n'y a alors rien de plus à faire, si le pas est implicite, il faut le renvoyer dans la pile AUX en stockant l'adresse de la cellule AUX dans la pile T.

On pourra donc dans tous les cas, exploiter une configuration unique de la pile T.

L'organigramme (figure VI-84) du traitement de FØR ... TØ ... STEP ... appelle un certain nombre de commentaires. Rappelons-nous que la pile T a l'aspect présenté à la figure VI-82. Le test du pas permet d'abord de détecter l'entrée dans une boucle infinie si le pas est nul (erreur 80), il permet ensuite l'aiguillage sur les tests de comparaison entre valeur finale et valeur initiale. Nous prenons ici comme hypothèse de travail, qu'il faut exécuter la boucle si :

valeur finale $\underset{\mathcal{E}(pas)}{\geqslant}$ valeur initiale , cette exécution de boucle correspond au cas représenté à partir de F507, où l'on insère dans le dernier champ de l'identificateur de boucle courant, le numéro de ligne de l'instruction qui suit l'instruction FØR.

Par contre, si valeur finale $\underset{\mathcal{E}(pas)}{<}$ valeur initiale , la boucle n'est pas valide, on élimine alors la cellule courante de la pile boucle, et on cherche l'instruction qui suit le dernier ordre de la boucle (NEXT) et qui doit être exécutée maintenant.

VI-14-4 - Traitement de NEXT

Après avoir appelé en mémoire centrale la piste qui contient la pile des boucles, on vérifie que le nom de la variable de boucle citée dans l'ordre NEXT, est le même que celui qui est enregistré dans l'identificateur de boucle courant. Un nom différent, signifie qu'il y a eu une entrée illégale dans la boucle.

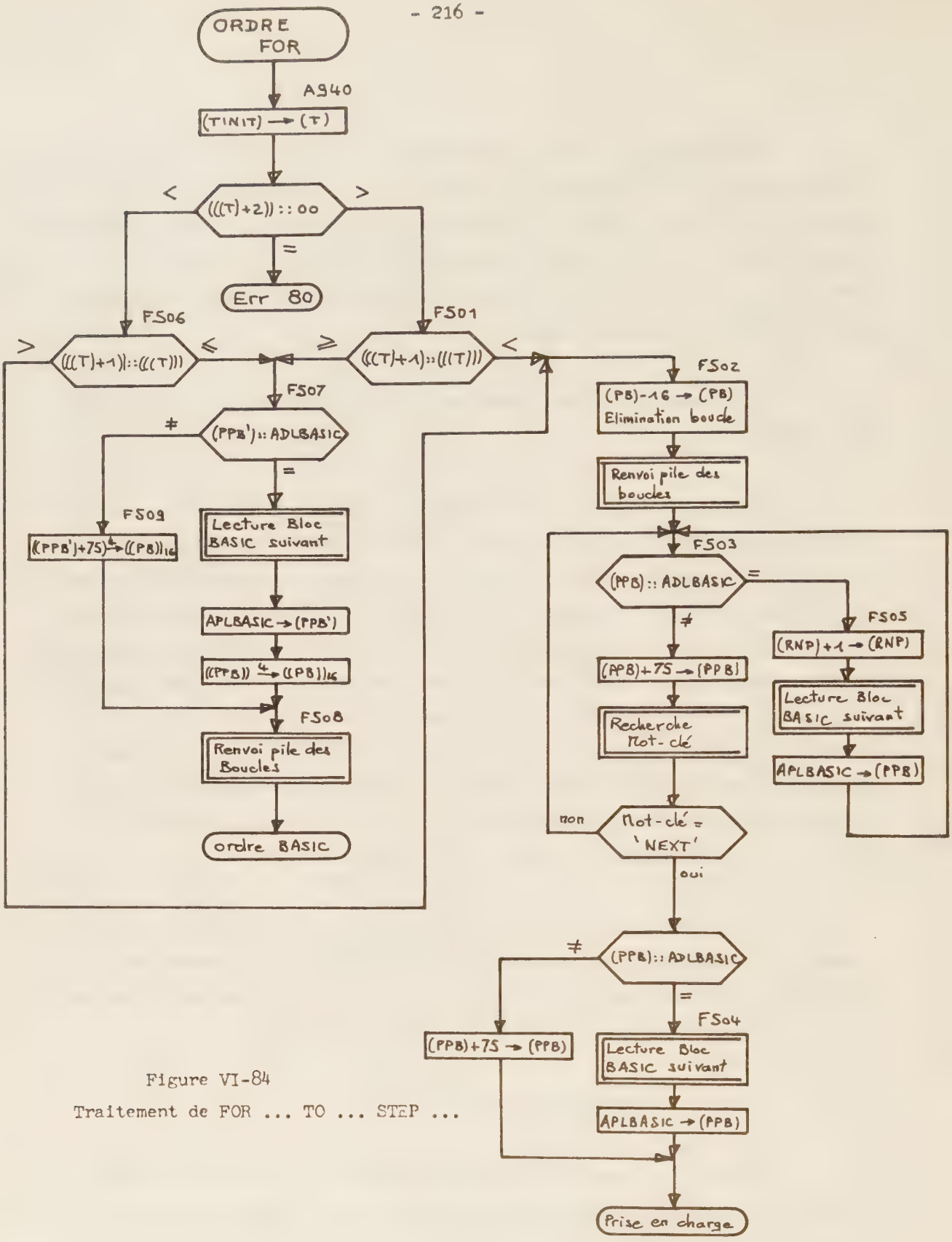


Figure VI-84

Traitement de FOR ... TO ... STEP ...

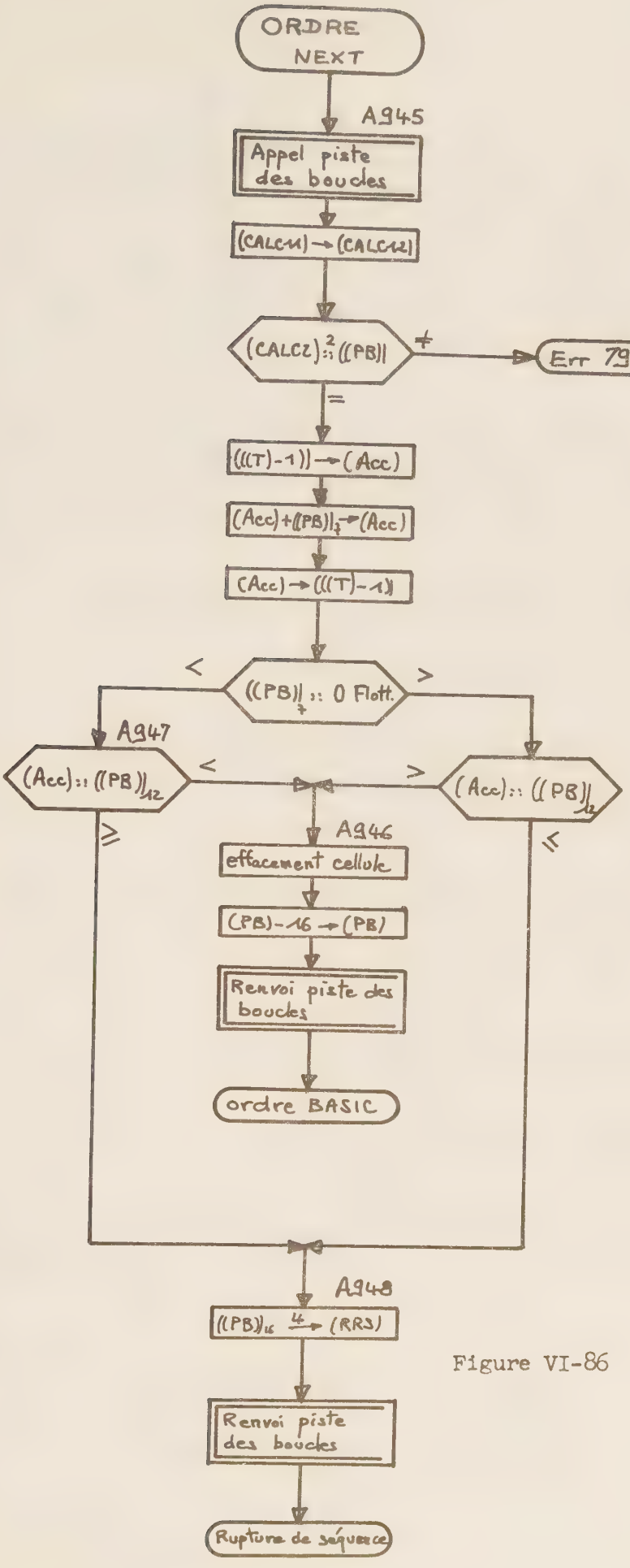

ORDRE NEXT
A945
Appel piste des boucles
(CALC1) → (CALC2)
(CALC2) :: ((PB))
+
Err 79
=
(((T)-1)) → (Acc)
(Acc) + ((PB))1 → (Acc)
(Acc) → (((T)-1))
<
((PB))1 :: 0 Flott.
>
A947
(Acc) :: ((PB))12
<
>
(Acc) :: ((PB))12
≥
≤
A946
effacement cellule
(PB) - 16 → (PB)
Renvoi piste des boucles
ordre BASIC
A948
((PB))16 4 → (RR3)
Renvoi piste des boucles
Rupture de séquence

Figure VI-86

On procède ensuite à l'addition du pas au contenu de la variable de boucle, que l'on compare à la valeur finale enregistrée dans l'identificateur. Le sens de la comparaison dépend du signe du pas. Quand on dépasse la valeur finale, il faut effacer la cellule courante, descendre la pile d'un cran, la renvoyer en tambour et terminer le traitement de façon normale par ordre BASIC.

Si la boucle n'est pas épuisée, il faut faire un branchement inconditionnel à l'instruction dont le numéro de ligne est enregistré dans la cellule courante, ce traitement est réalisé par la filière A948.

VI-15 - ORDRE IF ... THEN ... BRANCHEMENT CONDITIONNEL

VI-15-1 - Forme générale

× × × × IF relation THEN numéro de ligne

où la relation est composée de deux expressions séparées par un symbole de relation.

Les symboles de relation appartiennent au jeu suivant :

< , <= , = , <> , >= , >

Le symbole <> signifie "différent de".

VI-15-2 - Traitement

Nous nous arrangeons de telle sorte qu'à l'entrée dans le module d'interprétation de IF, la pile T ait la forme suivante :

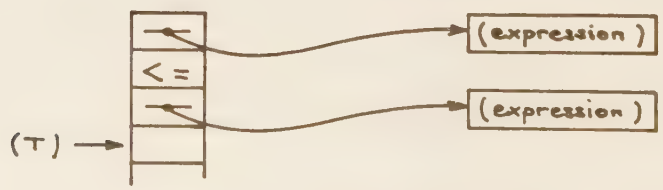


Figure VI-87

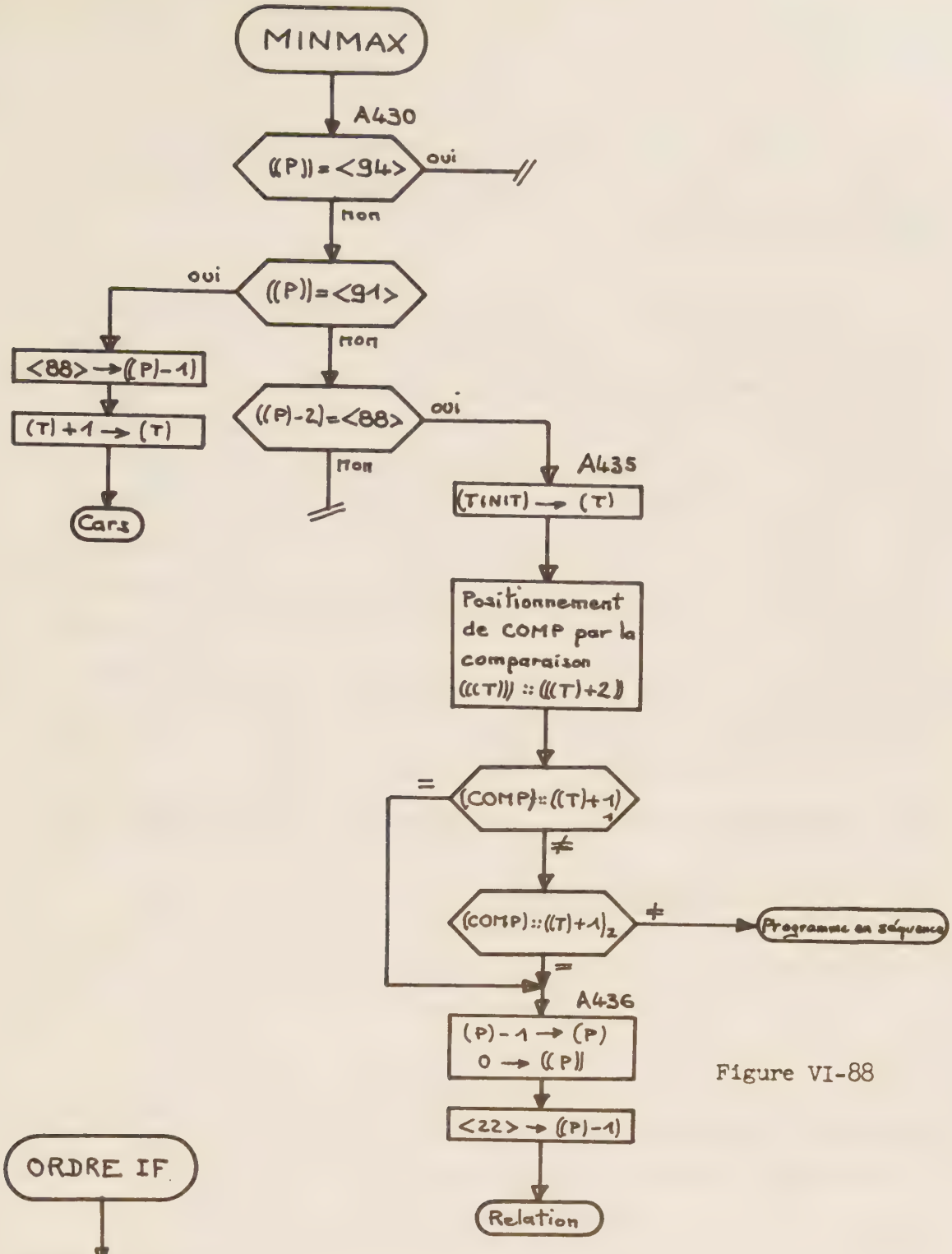


Figure VI-88

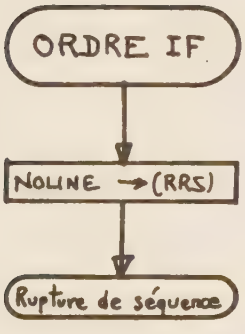


Figure VI-89

Les sous-relations étant des <minmax> du point de vue syntaxique, on dispose de la possibilité d'emploi de toutes les opérations arithmétiques.

Si l'on compare les 2 opérandes (le 1er avec le 2è), on obtient 3 résultats possibles $<$, $=$, $>$, il suffit de tester un octet CØMP qui contient l'un de ces résultats, en comparaison avec le ou les symboles de relation qui figurent dans la pile T. L'égalité indique alors que la relation est satisfaite et qu'il faut aller chercher le numéro de ligne cité après THEN pour aboutir finalement en ORDRE IF (figure VI-89) et faire le branchement inconditionnel. La non-égalité indique que la relation n'est pas satisfaite, on peut alors abandonner l'instruction en cours et aller chercher la suivante, c'est le passage au programme en séquence (figure VI-88). D'autre part, dans MINMAX, on avance la pile T d'un cran quand on détecte un symbole de relation <91> au sommet de la pile d'analyse.

VI-16 - ORDRES GØSUB ET RETURN - APPELS ET RETOURS DE SOUS-PROGRAMMES

VI-16-1 - Forme générale

× × × × GØSUB numéro de ligne
× × × × RETURN

On définit en fait des sous-programmes internes. La séquence qui constitue le sous-programme commence par l'instruction dont l'étiquette est citée dans l'instruction GØSUB, elle se termine par l'ordre RETURN qui renvoie la commande à l'instruction qui suit l'appel .

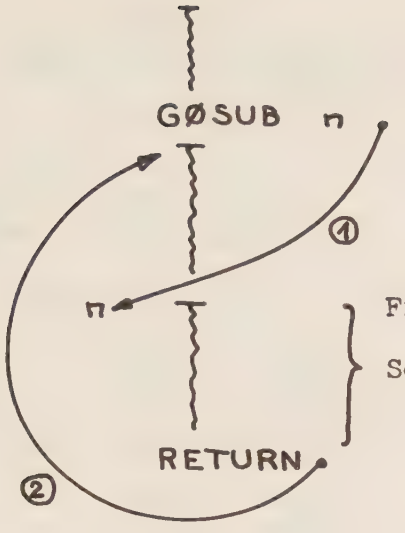


Figure VI-90

Sous-programme interne

VI-16-2 - Traitement

Le traitement de ces instructions doit tenir compte de la possibilité d'encastrement des sous-programmes. On utilise pour cela une pile SP qui sera renvoyée en tambour en dehors du traitement des ordres GØSUB et RETURN.

Quand on rencontre un ordre GØSUB, on monte la pile SP d'un cran, on loge dans la cellule du sommet le numéro de la ligne qui suit l'appel. RETURN a pour effet de récupérer le numéro de la ligne de renvoi, de descendre la pile SP d'un cran.

L'erreur 83 survient quand la pile dépasse le maximum correspondant à 20 sous-programmes encastrés. NØLINE est chargé dans ENTIER (figure VI-74), car IND9 est positionné à 1 par le mot-clé GØSUB.

Dans ØRDRE RETURN (figure VI-92), le test (PSP)::(BUTEE) est implanté pour vérifier qu'il y a eu un appel du sous-programme dont on est en train de sortir.

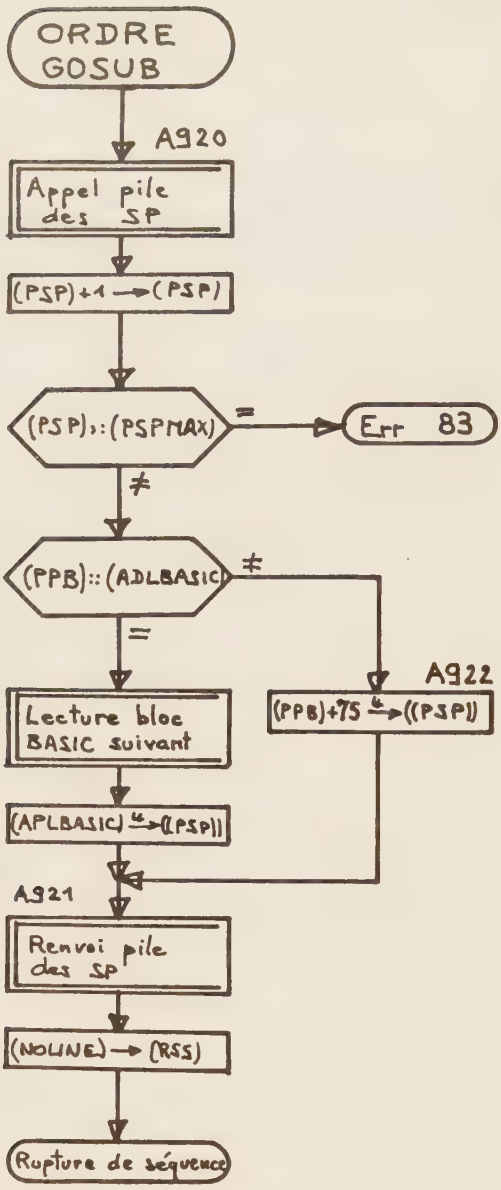


Figure VI-91

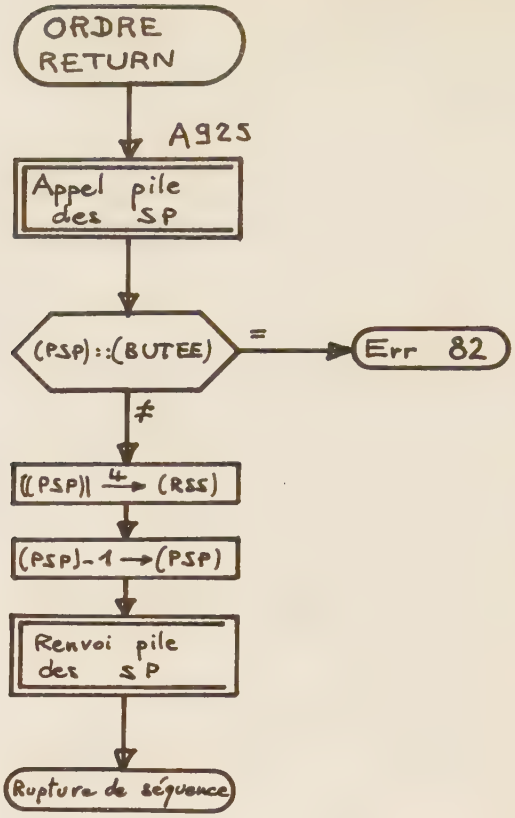


Figure VI-92

VI-17 - ORDRE DEF - DEFINITION DE FONCTION INTERNE

L'ordre DEF a en fait déjà été étudié au paragraphe VI-7 - Traitement des fonctions.

VI-18 - INSTRUCTIONS NON EXECUTABLES - ORDRES DATA ET REM

Il existe en BASIC des instructions non-exécutables, c'est-à-dire ne donnant pas lieu à une action sur le déroulement du programme.

La première de ces instructions est l'instruction DATA, dont le seul rôle est de conserver en mémoire des constantes numériques dont on a besoin dans le programme et qu'on exploite par les ordres READ.

La seconde, l'instruction REM correspond à l'introduction d'une ligne de commentaires (remarques) dans le programme.

Les DATA ayant déjà été pris en charge par ailleurs, le traitement de ces instructions se résume au passage à l'instruction suivante quand on détecte un des mots-clés DATA et REM. On introduit dans la table de branchement aux mots-clés, un branchement inconditionnel à la routine PASSAGE EN SEQUENCE (figure VI-64).

VI-19 - INSTRUCTION DE RESERVATION DE TABLEAU - ORDRE DIM

VI-19-1 - Forme générale

× × × × DIM identificateur de tableau (entier), ...
× × × × DIM identificateur de tableau (entier, entier), ...

Il n'y a pas de virgule après le dernier tableau dimensionné.

VI-19-2 - Traitement

Lors des explications partant sur les variables indicées (figure VI-6), nous avons vu comment on crée une table d'identificateurs matriciels (voir figure VI-16)

En fait les cellules ne sont pas complètement remplies, il n'y a encore à ce niveau que le nom du tableau, il faut maintenant insérer les dimensions telles qu'elles apparaissent dans l'ordre DIM, ainsi que le numéro de la piste où on loge le tableau.

On sait que les quantités qui figurent entre parenthèses derrière l'identificateur du tableau sont des entiers, il y a donc un travail d'interprétation à faire au niveau de la routine ENTIER, ce travail consiste à changer la représentation étendue en condensée, à vérifier que la quantité obtenue n'est pas supérieure à 140 qui représente la borne supérieure d'une dimension et à remettre DEFCT à zéro pour permettre le traitement de la seconde dimension éventuelle.

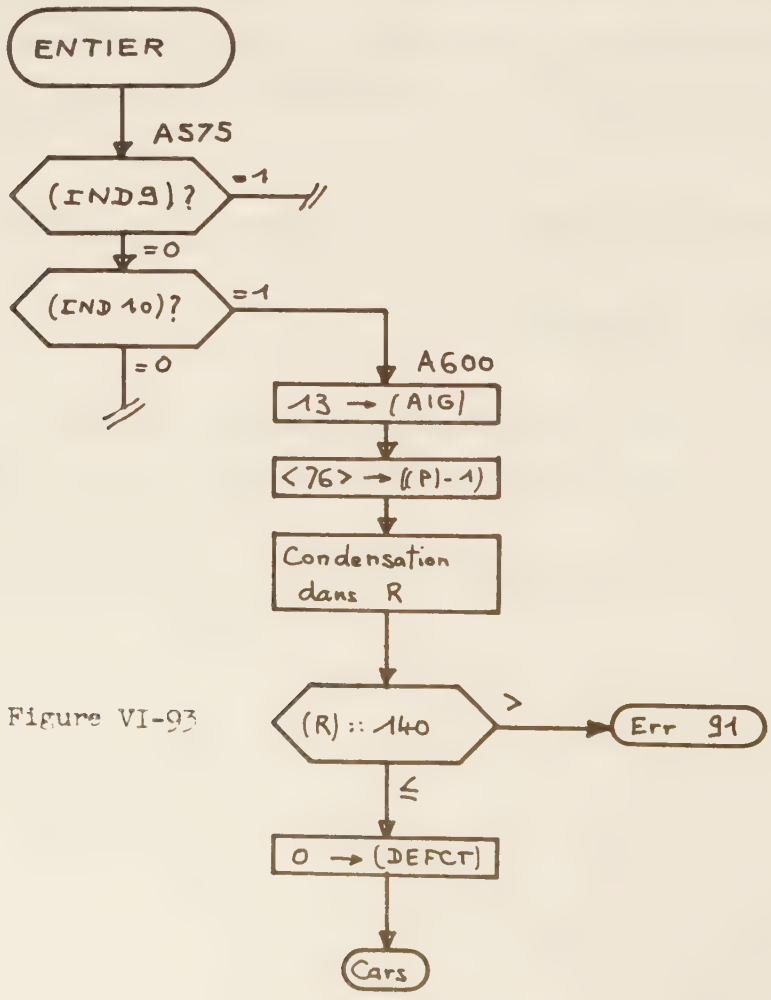


Figure VI-93

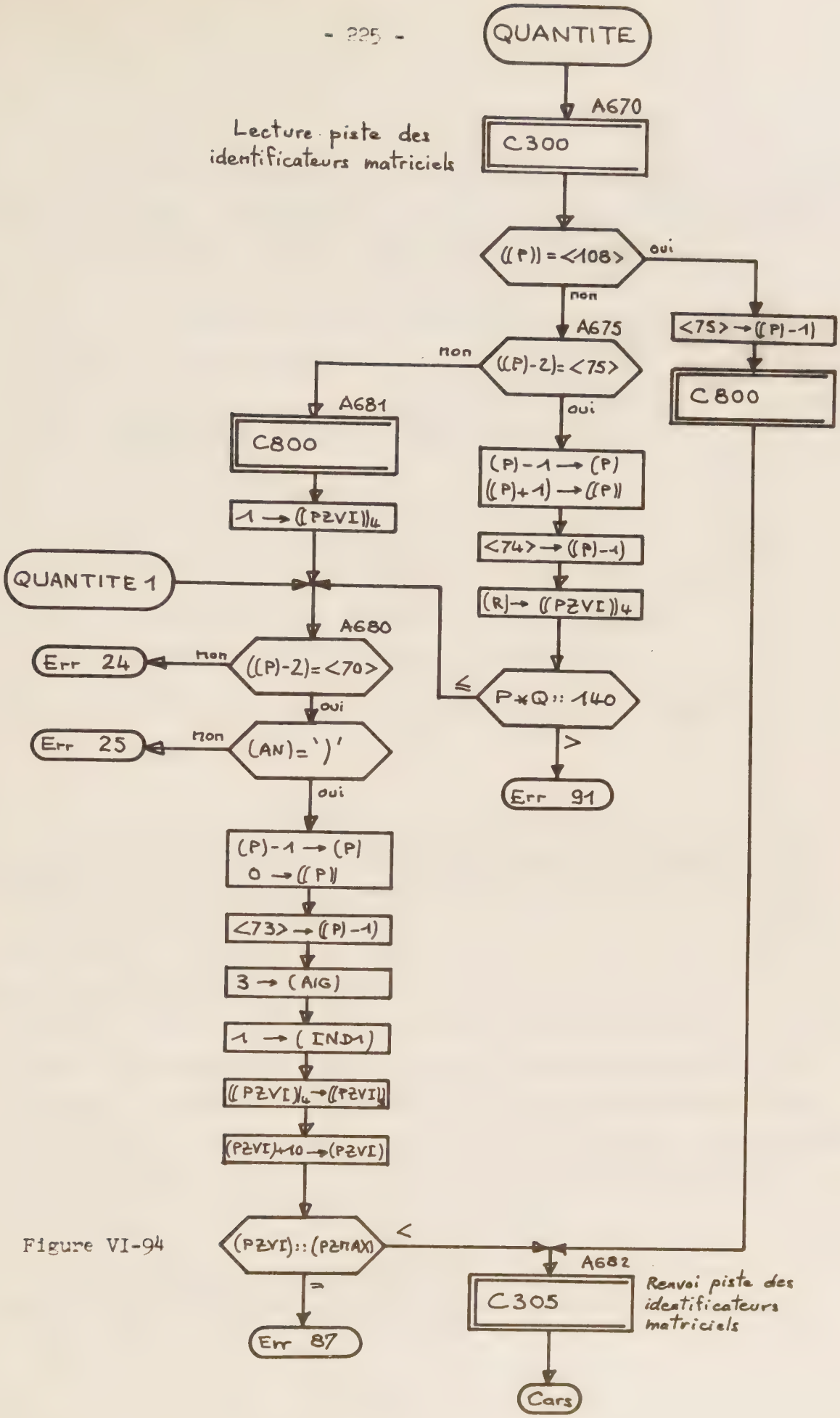

QUANTITE
A670
C300
Lecture piste des identificateurs matriciels
((P)) = <108>
oui
non
A675
((P)-2)= <75>
non
oui
<75> → ((P)-1)
C800
A681
C800
(P)-1 → (P)
((P)+1) → ((P))
<74> → ((P)-1)
(R) → ((PZVI))4
1 → ((PZVI))4
QUANTITE 1
A680
Err 24
non
((P)-2)= <70>
oui
Err 25
non
(AN)= ')'
oui
P*Q :: 140
≤
>
Err 91
(P)-1 → (P)
0 → ((P))
<73> → ((P)-1)
3 → (AIG)
1 → (IND1)
((PZVI))4 → ((PZVI))
(PZVI)+10 → (PZVI)
(PZVI) :: (PZMAX)
<
=
A682
C305
Renvoi piste des identificateurs matriciels
Err 87
Cars

Figure VI-94

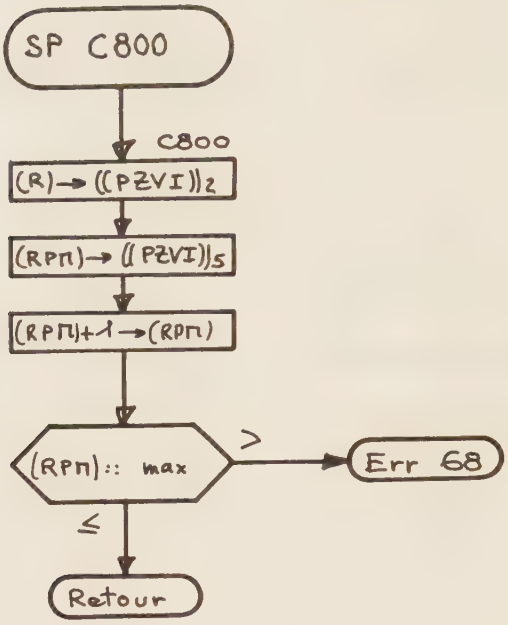


Figure VI-95

C'est évidemment dans QUANTITE qu'on traite les dimensions spécifiées, on appelle d'abord la piste des identificateurs matriciels (figure VI-94), la présence de <108> dans ((P)) signifie qu'il y a définition d'un tableau à 2 dimensions, dont on traite la première par C800 (figure VI-95) en prenant dans R la dimension sous forme condensée et en la chargeant dans le second champ de l'identificateur, puis en entrant le contenu du registre piste matrice (RPM) dans le 4è champ, il y a incrémentation de RPM et test d'encombrement.

S'il n'y a pas <108> , il y a 2 solutions : ou bien le tableau n'a qu'une dimension que l'on traite par C800 (filière A681) et en prenant la seconde dimension égale à 1 ; ou il y a 2 dimensions, la 1ère ayant déjà été traitée, on introduit dans le IIIème champ le contenu de R, on vérifie que le produit P * Q des dimensions ne dépasse pas 140.

Dans QUANTITE1 (A680), il y a recopie des dimensions spécifiées dans les champs correspondant aux dimensions courantes, puis incrémentation du pointeur PZVI avec vérification de non-débordement.

On termine enfin par le renvoi en tambour de la table des identificateurs matriciels.

VI-20 - INSTRUCTION DE SORTIE - ORDRE PRINT

VI-20-1 - Forme générale

× × × × PRINT expression, expression, ...

× × × × PRINT "libellé" ; expression ; ...

× × × × PRINT "libellé" ; expression ; "libellé", ...

× × × × PRINT toute combinaison de libellés et/ou d'expressions

× × × × PRINT

Il y a donc deux sortes d'opérandes : les expressions et les libellés alphanumériques séparés par deux sortes de délimiteurs : les virgules ou les points-virgules. Le délimiteur final peut être une virgule, un point-virgule ou rien.

VI-20-2 - Traitement

La mise en page dépend essentiellement des délimiteurs employés. L'emploi du délimiteur virgule permet de définir des champs d'impression de 15 positions, une ligne de télescripteur étant composée de 72 positions, on a en fait 4 champs de 15 et 1 champ de 12 positions.

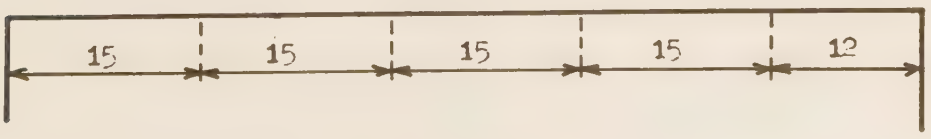


Figure VI-96

Si dans un ordre PRINT, il y a une série d'expressions séparées par des virgules, la valeur de chacune d'entre-elles s'imprime dans chaque zone avec cadrage à gauche, sauf peut-être la cinquième de chaque ligne si elle ne tient pas sur 12 positions, elle est alors renvoyée dans le 1er champ de la ligne suivante.

L'emploi du libellé entre guillemets suivi de la virgule, a pour effet de faire imprimer le libellé (son cadrage gauche est variable puisqu'il dépend du délimiteur précédant) et de faire imprimer l'opérande suivant (libellé ou valeur d'une expression) cadré à gauche dans la zone suivante.

Il est possible d'utiliser des champs de longueur variable selon l'encombrement imposé par la valeur d'une expression ou par la longueur d'un libellé, on emploie dans ce but, le délimiteur point-virgule. On dit alors que l'impression est condensée.

On a un champ de 6 positions pour une valeur numérique de 1 à 3 caractères
On a un champ de 9 positions pour une valeur numérique de 4 à 6 caractères
On a un champ de 12 positions pour une valeur numérique de 7 à 9 caractères
On a un champ de 15 positions pour les autres valeurs.

Pour les libellés alphanumériques, l'utilisation du point-virgule a pour effet de faire commencer le champ suivant à la position qui suit le dernier caractère imprimé du texte.

Enfin le délimiteur final s'il est vide, commande un passage à la ligne, c'est vrai en particulier pour l'ordre PRINT sans opérandes.

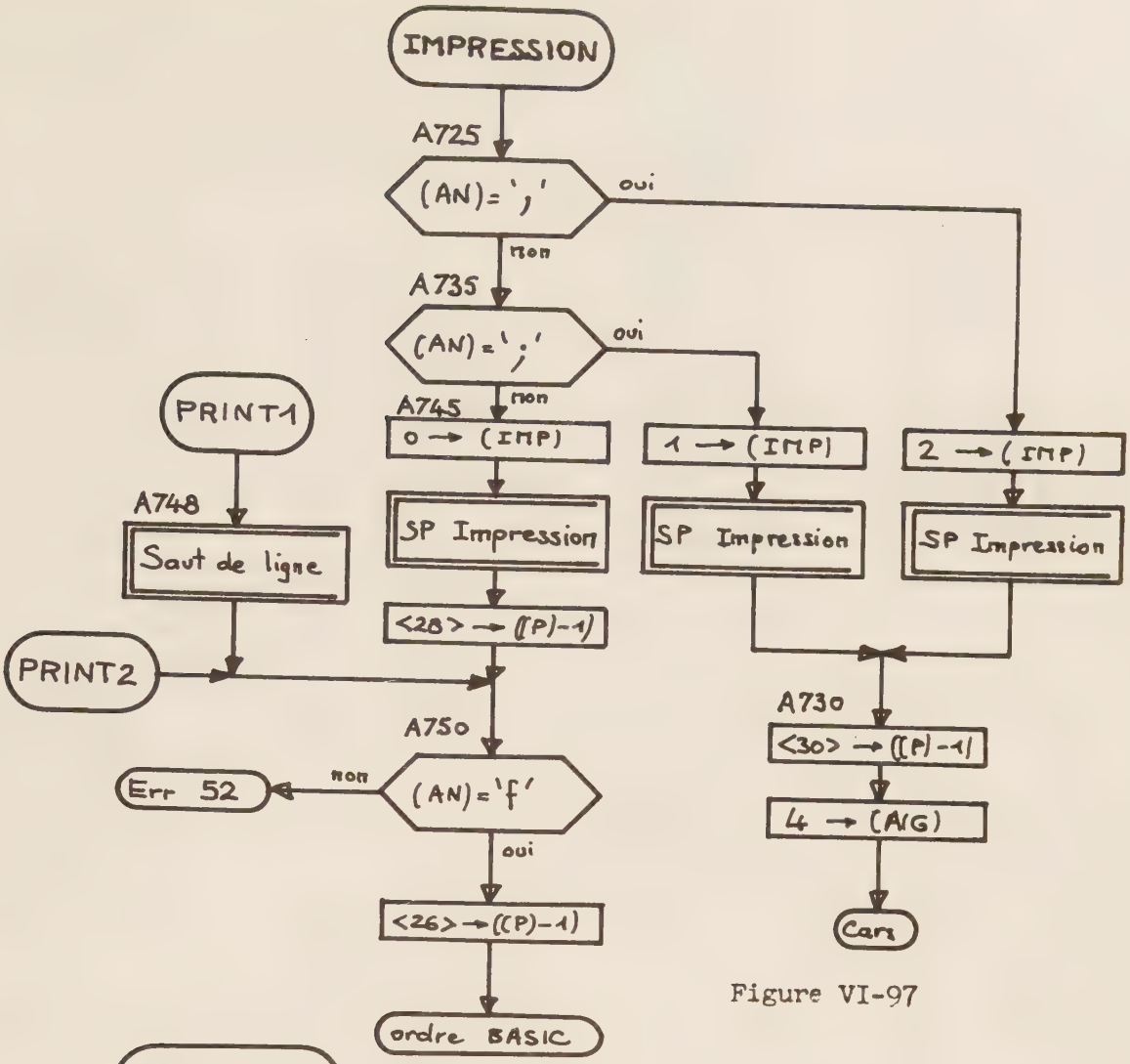


Figure VI-97

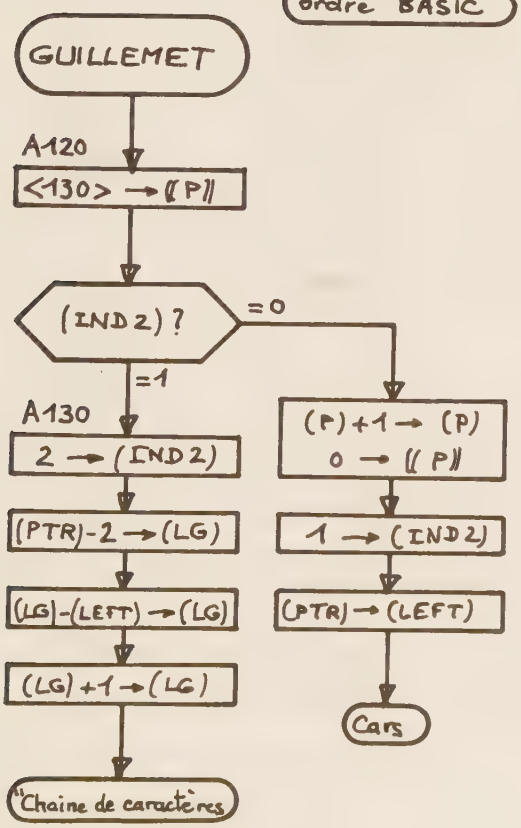


Figure VI-98

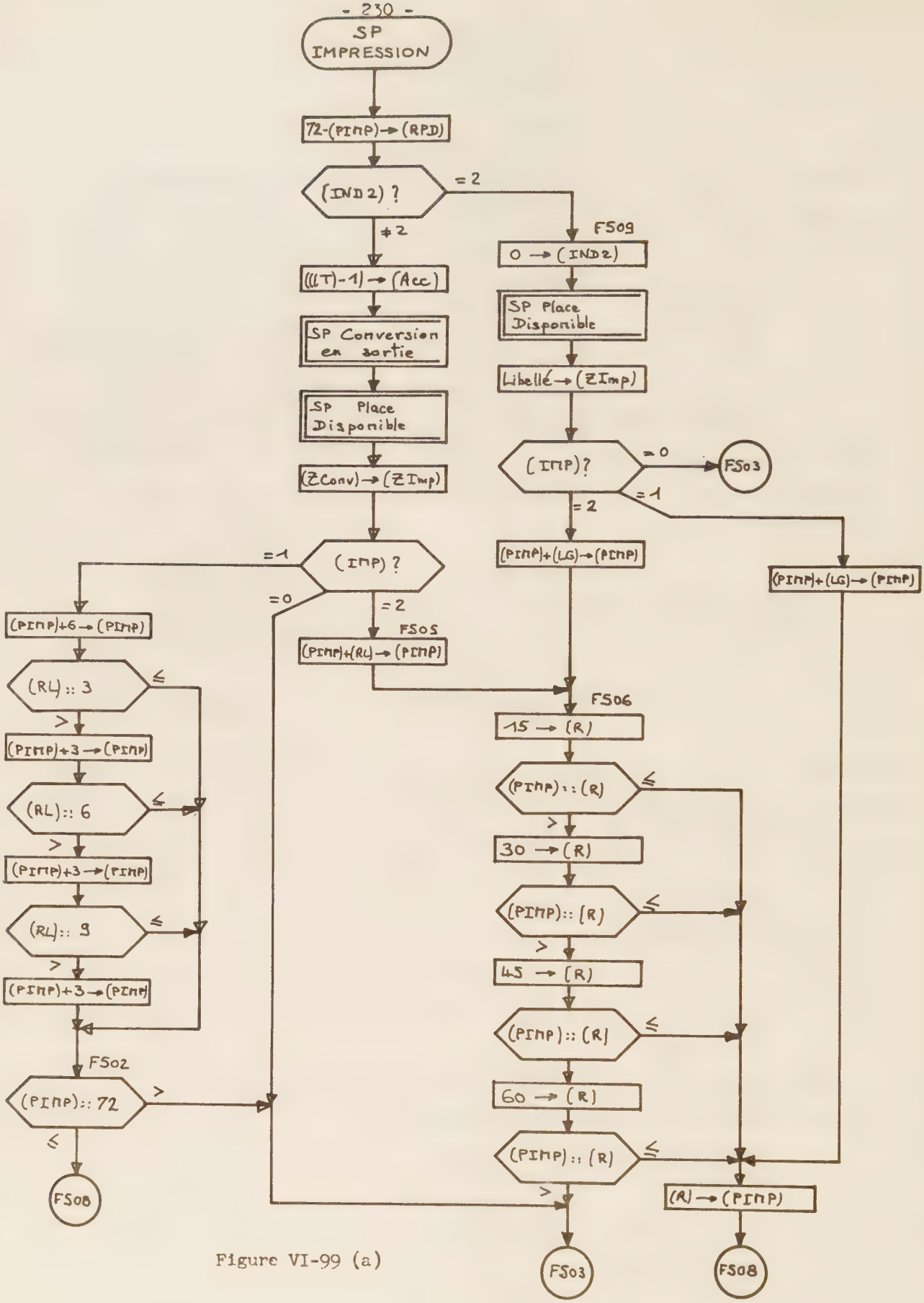


Figure VI-99 (a)

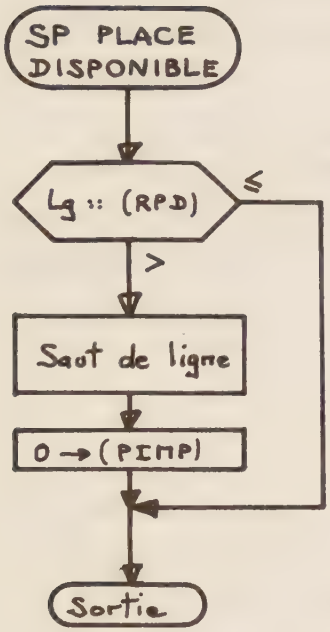


Figure VI-100

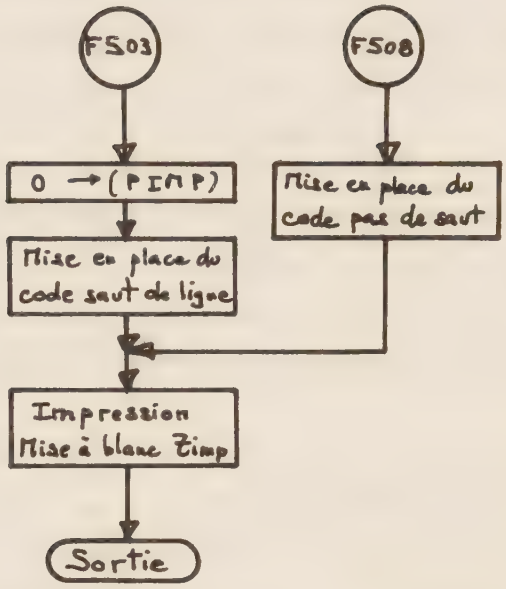


Figure VI-99 (b)

Le module IMPRESSIØN (figure VI-97) montre l'utilisation d'un seul sous-programme d'impression piloté par l'indicateur IMP. Cet indicateur contient 0 si le délimiteur est vide ; 1 si le délimiteur est un point-virgule, 2 si le délimiteur est une virgule. L'entrée PRINT1 commandant un saut de ligne correspond au cas où le mot-clé PRINT figure seul dans l'instruction.

La routine GUILLEMET (figure VI-98) permet d'enregistrer l'adresse gauche du libellé à la détection des guillemets d'ouverture. On passe l'indicateur IND2 à 2 dans A130 à la détection des guillemets de fermeture et on calcule la longueur du libellé.

Le sous-programme impression (figure VI-99) commence par le calcul de la place disponible sur la ligne en cours, PIMP contenant le numéro de la position qui débute le champ immédiatement disponible.

Le test d'IND2 permet de séparer les traitements des libellés et des valeurs numériques. Dans les deux cas (après conversion externe des valeurs numériques), on appelle un sous-programme qui vérifie que la place disponible est suffisante pour permettre l'impression sans débordement, si ce n'est pas le cas, il y a saut de ligne avec remise à zéro de PIMP.

Après la mise en place de la chaîne à imprimer dans la zone impression (ZImp), il y a incrémentation de PIMP (variable selon le contenu de IMP) et impression sans saut de ligne s'il n'y a pas dépassement de PIMP, ou impression avec saut de ligne dans le cas contraire, qui correspond également au cas où le délimiteur final est vide.

Le sous-programme de conversion en sortie n'est pas le sous-programme EXTFLO (fourni par le constructeur) qui ne correspond pas à une mise en page automatique du genre de celle dont nous avons besoin. On trouvera en annexe des détails sur le sous-programme réalisé.

VI-21 - INSTRUCTION DE FIN D'EXECUTION - ORDRES STOP ET END

Ces instructions qui ne comportent que le mot-clé permettent d'interrompre l'exécution du programme BASIC. Il s'agit alors d'interruption normale, l'interruption anormale étant liée à la détection d'une erreur d'interprétation.

Remarquons que l'ordre END qui doit conclure un programme peut être inséré en plusieurs exemplaires dans le programme, il remplace alors l'ordre STOP dont on pourrait se passer dans ces conditions.

VI-22 - INSTRUCTIONS MATRICIELLES - ORDRES MAT, MATINPUT, MATREAD, MATPRINT

VI-22-1 - Généralités

Nous réunissons dans un même paragraphe toutes les instructions qui travaillent directement sur des noms de tableaux, ceci parce qu'en fait on peut considérer ces instructions indépendamment de toutes les autres, dans la mesure où toutes les instructions que nous avons vues

jusqu'à maintenant constituent un ensemble homogène parfaitement utilisable. Cet ensemble peut constituer une sorte de MINIBASIC, sous-ensemble de BASIC.

Néanmoins ce sous-langage étant privé du sous-langage matriciel n'affiche plus la supériorité que lui donne l'équipement des fonctions et des opérations matricielles, et redevient un langage évolué certes, mais de faible puissance, comme le MINIFORTRAN par exemple.

Il était donc intéressant dans le cadre de cette étude, de réaliser le traitement des instructions matricielles.

VI-22-2 - Instruction de lecture de matrice - Ordre MATREAD

VI-22-2-1 - Formes générales

× × × × MATREAD nom de tableau
× × × × MATREAD nom de tableau (expression)
× × × × MATREAD nom de tableau (expression, expression)

Dans le premier cas, le nombre d'éléments lus correspond aux dimensions courantes du tableau considéré.

Dans le second cas, le tableau doit n'avoir qu'une dimension, le nombre d'éléments lus correspond à la valeur de l'expression qui devient la nouvelle dimension courante. Le raisonnement est identique dans le troisième cas où le tableau doit avoir deux dimensions.

Dans une même instruction MATREAD, on peut mélanger les 3 types définis ci-dessus en les séparant par des virgules, le dernier opérande étant suivi de blancs.

Cette instruction permet de lire dans les lignes DATA courantes, tous les éléments d'une matrice. La lecture se fait ligne par ligne.

VI-22-2-2 - Traitement

Le traitement des nouvelles dimensions a déjà été étudié au paragraphe VI-6-5, nous nous préoccupons simplement du module de lecture des matrices inclus dans la routine TABLEAU.

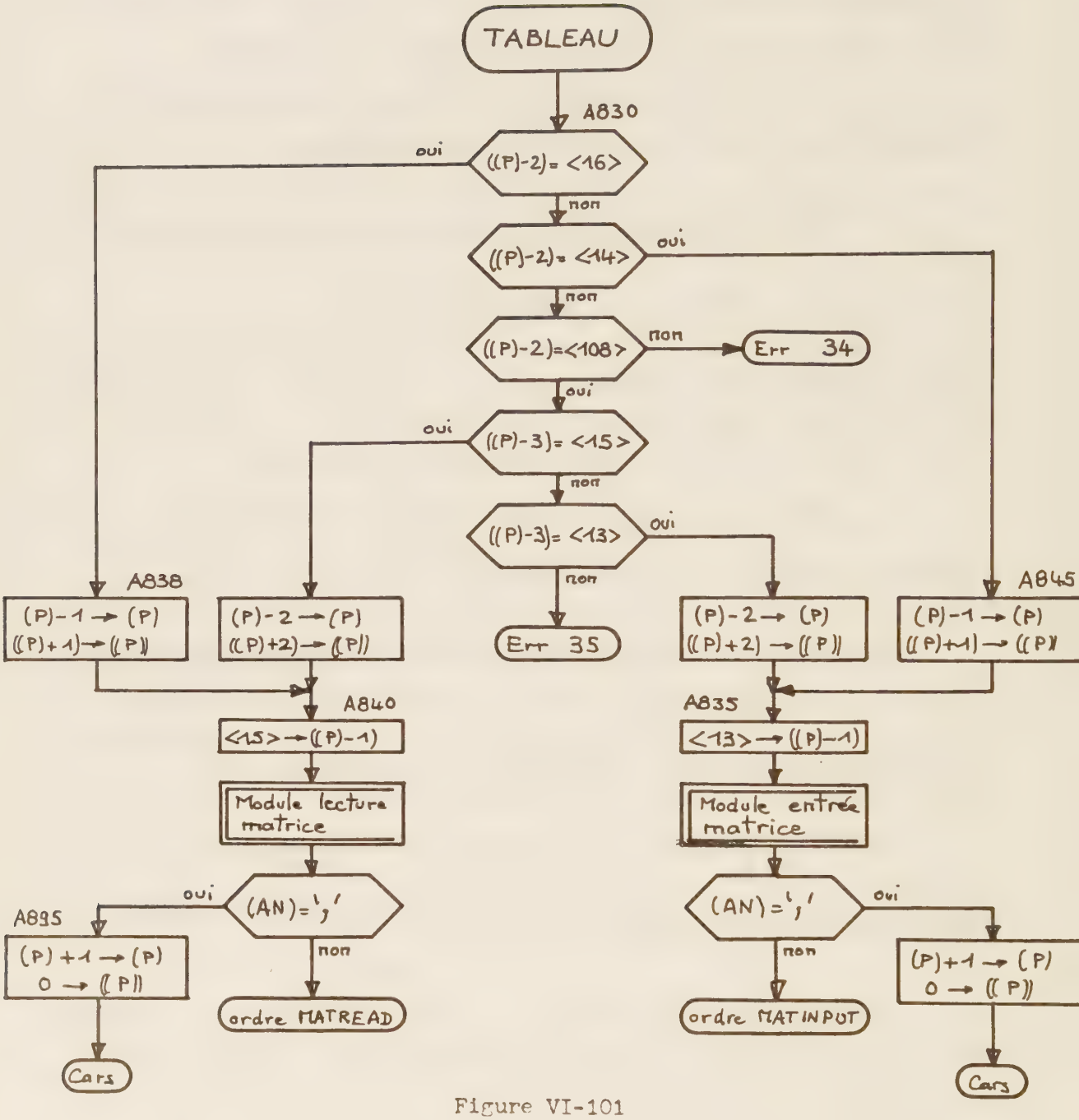


Figure VI-101

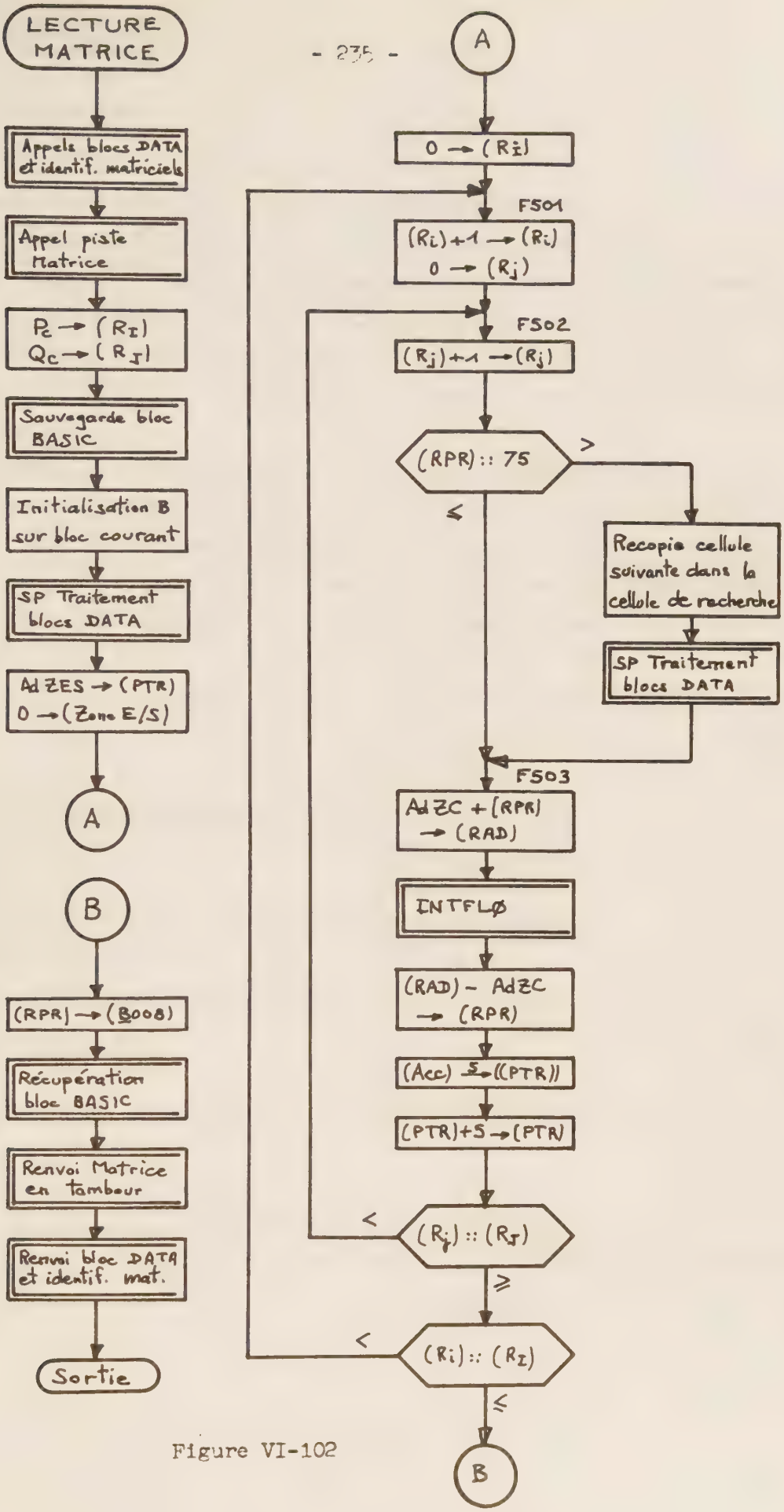


Figure VI-102

Pour lire une matrice (figure VI-102), il faut avoir en mémoire le bloc de renseignements sur les DATA, l'identificateur matriciel concerné, on peut alors appeler la matrice en mémoire. Après différents traitements relatifs à la manipulation des DATA, on remet à zéro la zone d'entrée-sortie qui reçoit la matrice, après avoir chargé un pointeur avec l'adresse de tête de la matrice.

On peut maintenant lire dans les lignes DATA, les éléments un par un, convertir le résultat et l'affecter correctement à l'aide du pointeur PTR, ces opérations s'effectuent dans une double boucle, sous le contrôle de registres-indicés (R_i et R_j) initialisés à 1 (0 + 1) et atteignant les dimensions courantes (P_c et Q_c) chargées dans R_I et R_J.

Après diverses mises à jour, on peut renvoyer la matrice en tambour.

VI-22-3 - Instruction d'entrée au clavier de matrice Ordre MATINPUT

VI-22-3-1 - Formes générales

Elles sont du même type que celles employées avec MATREAD, on ne change que le mot-clé.

On a les mêmes considérations que précédemment quant aux dimensions.

Le but de cette instruction est d'entrer au clavier alphanumérique tous les éléments constituant une matrice donnée. Les éléments sont entrés ligne à ligne.

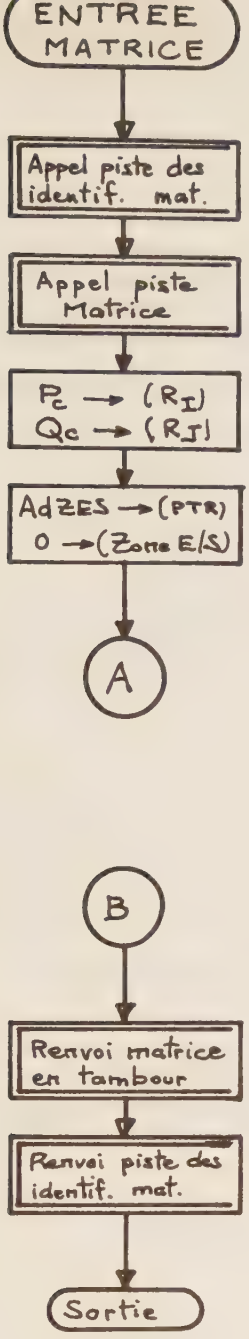


Figure VI-103

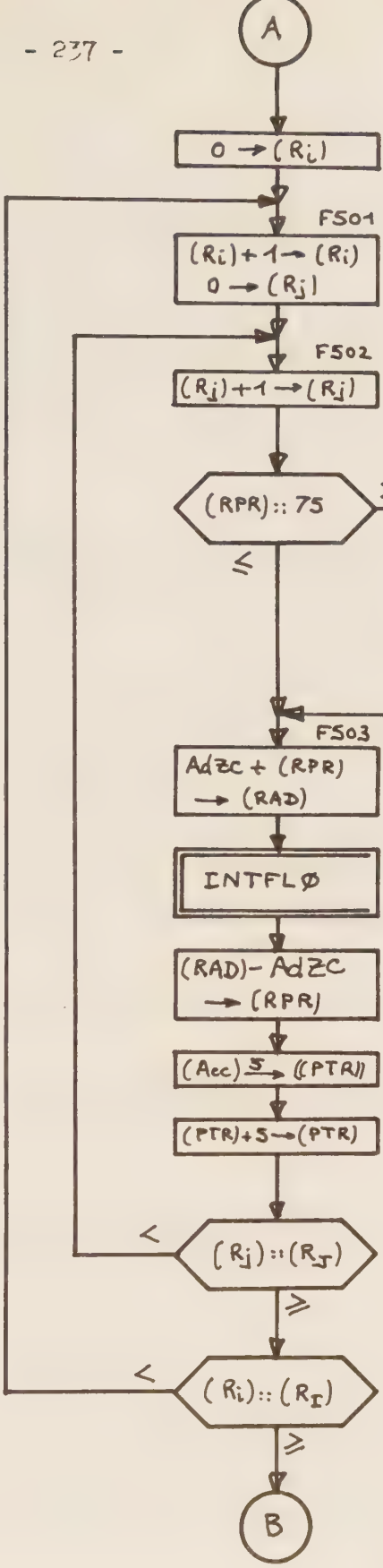
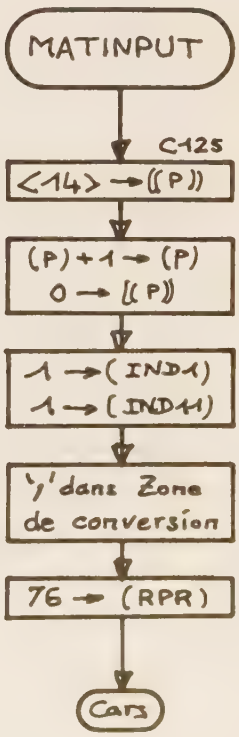


Figure VI-104



VI-22-3-2 - Traitement

Comme pour l'ordre INPUT, on utilise une zone de conversion dans laquelle le sous-programme de frappe des données numériques dépose les caractères tapés sur le clavier, et qui constituent la chaîne terminale transformée en quantité en flottant interne par le S/P de conversion INTFLØ .

Dans MATINPUT (figure VI-104), on dispose une virgule en bout de la zone de conversion et on initialise (RPR) à 76, de façon à appeler la première fois le S/P de frappe.

On peut frapper plusieurs données séparées par des virgules et qui sont affectées en séquence à des éléments successifs de la matrice, ce qui permet de frapper toute une ligne d'un coup et de ne rappeler le S/P de frappe que pour une autre ligne, c'est pourquoi (RPR) peut être inférieur à 75 (figure VI-103).

VI-22-4 - Instruction d'impression de matrice - Ordre MATPRINT

VI-22-4-1 - Forme générale

```
× × × ×  MATPRINT nom de matrice
× × × ×  MATPRINT nom de matrice, nom de matrice ...
× × × ×  MATPRINT nom de matrice ; nom de matrice ...
```

Si derrière le nom de la matrice il n'y a pas de délimiteur, ou si le délimiteur est une virgule, les éléments de la matrice sont imprimés 5 par 5 dans des champs de taille normale (15 positions) avec interligne simple jusqu'à épuisement de la ligne courante de la matrice, il y a alors saut d'un interligne double avant d'entreprendre l'impression de la ligne suivante.

```
0       1       2       3       4
5       6       7
```

Figure VI-105

```
8       9       10      11      12
13      14      15
```

L'exemple précédent montre l'impression "large", d'une matrice (2 x 8).

Si le délimiteur est un point-virgule, l'impression a lieu dans des champs "condensés", le nombre d'éléments figurant sur une ligne d'imprimante est alors variable.

```
0  1  2  3  4  5  6  7
8  9  10 11 12 13 14 15
```

Figure VI-106

La figure VI-106 montre l'impression "condensé" de la même matrice.

VI-22-4-2 - Traitement

Quand on arrive dans MATPRINT1, le code syntaxique du délimiteur figure au sommet de la pile P, si c'est <31> qui correspond au point-virgule, on positionne IMP à 1 avant d'appeler le sous-programme d'impression des matrices, sinon l'impression a lieu en "large" que le délimiteur soit une virgule ou vide (en fait le caractère spécial de fin).

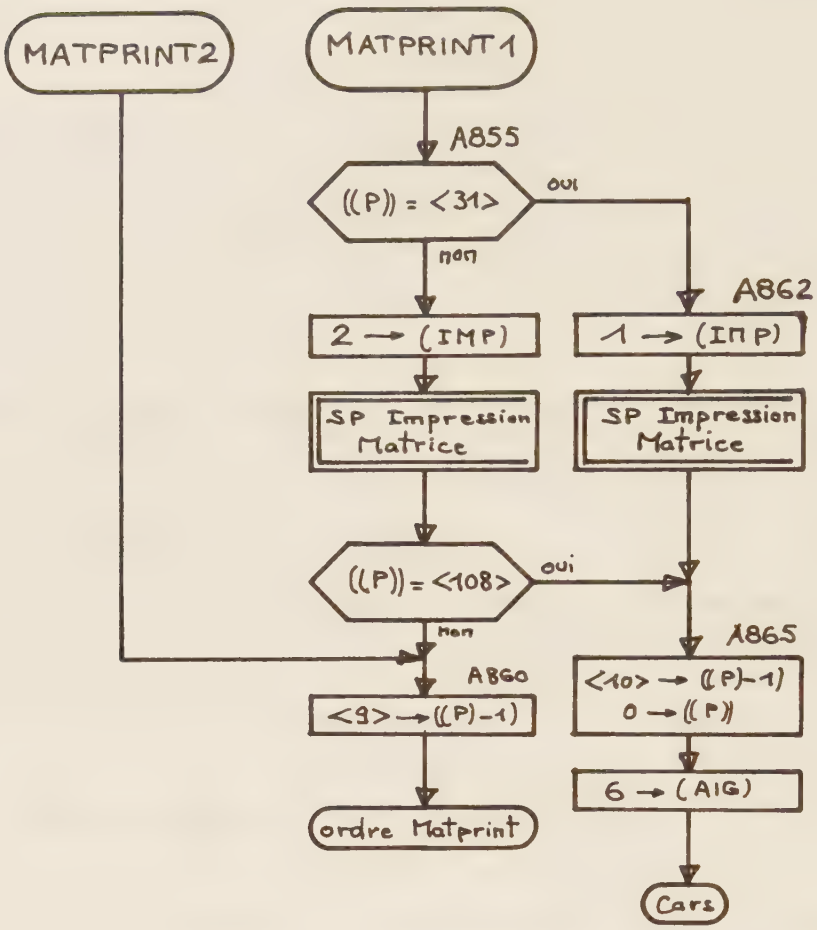


Figure VI-107

Le sous-programme d'impression des matrices appelle peu de commentaires, il faut simplement prévoir l'utilisation d'une cellule AUX dans laquelle on loge l'élément à imprimer pour rester compatible avec le S/P Impression (paragraphe VI-20). Remarquons qu'il y a au total un interligne quadruple qui sépare la dernière ligne imprimée de la matrice et la ligne qui sera éventuellement imprimée dans la suite du programme BASIC. Il faut aussi remettre à zéro PIMP le pointeur impression, ce qui permettre à l'impression future de s'effectuer dans le premier champ.

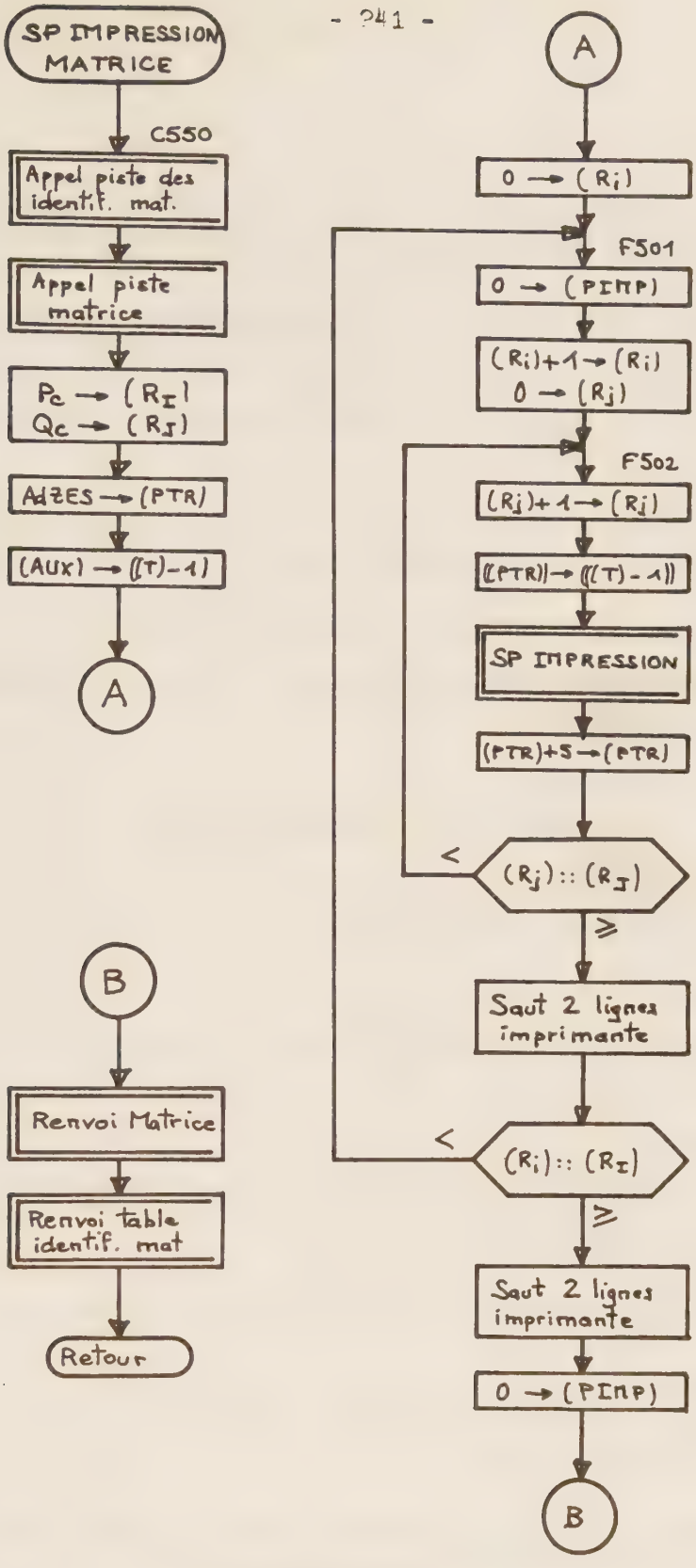


Figure VI-108

VI-22-5 - Autres Instructions matricielles - Ordre MAT

VI-22-5-1 - Fonctions initialisatrices

VI-22-5-1-1 - Types de fonctions initialisatrices

Il existe trois types de fonctions initialisatrices :

ZER qui permet la mise à zéro de tous les éléments d'une matrice .

CØN qui permet la mise à 1 de tous les éléments d'une matrice.

IDN qui permet à la matrice de devenir une matrice identité, ou encore matrice unité.

VI-22-5-1-2 - Formes générales

Elles sont les mêmes pour les 3 types.

× × × × MAT nom de matrice = ZER
Les dimensions employées sont les dimensions courantes de la matrice.

× × × × MAT nom de matrice = ZER (expression)
Il s'agit d'une matrice colonne dont la dimension est donnée par la valeur de l'expression.

× × × × MAT nom de matrice = ZER (expression, expression)

Les dimensions de la matrice sont redéfinies par les 2 expressions.

Dans le cas IDN, la matrice citée doit être exclusivement carrée, la 2ème forme n'est donc pas applicable.

VI-22-5-1-3 - Traitement

Le traitement des nouvelles dimensions, le cas échéant, est réalisé par le module redimensionnement déjà étudié.

Il est pratique, à la reconnaissance de ces fonctions (figure VI-109) de ranger leur nom dans une mémoire réservée à cet effet, NFI (nom fonction initialisatrice) exploitable par le module initialisation, lequel est inséré à la fin de la routine INITIALISATIØN/FØNCTIØN INITIALISATRICE (figure VI-110).

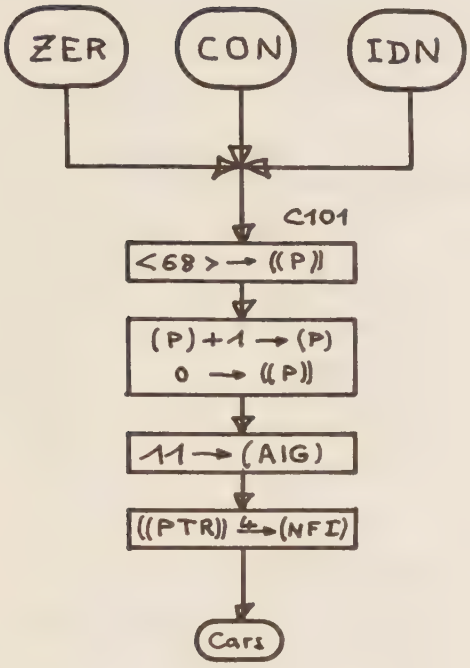


Figure VI-109

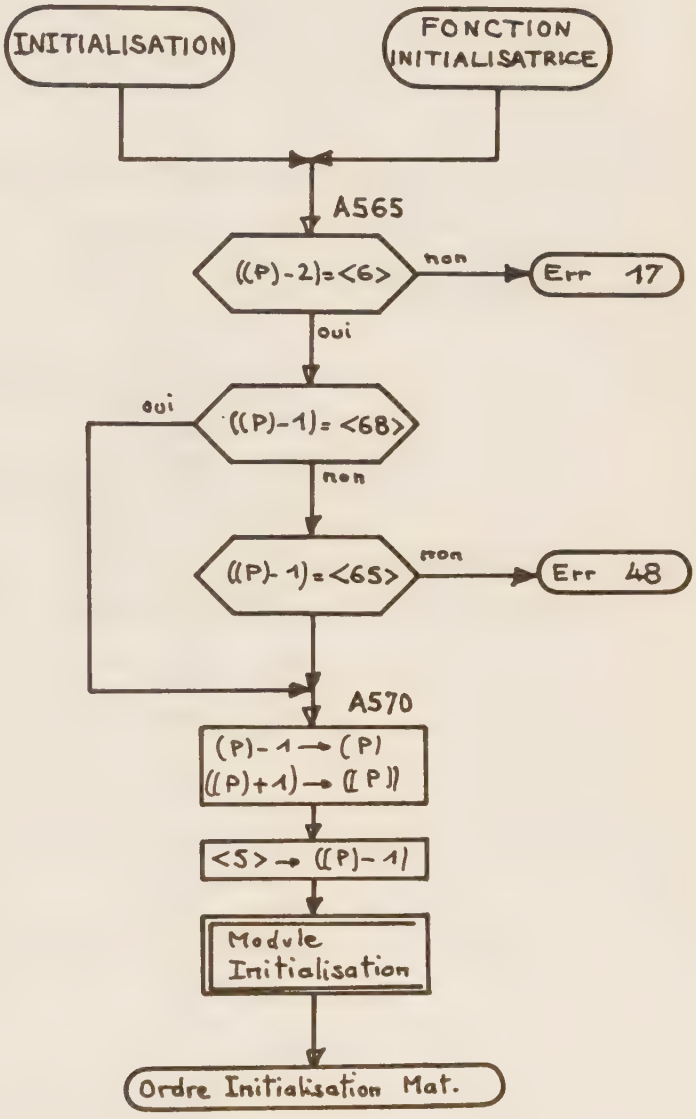


Figure VI-110

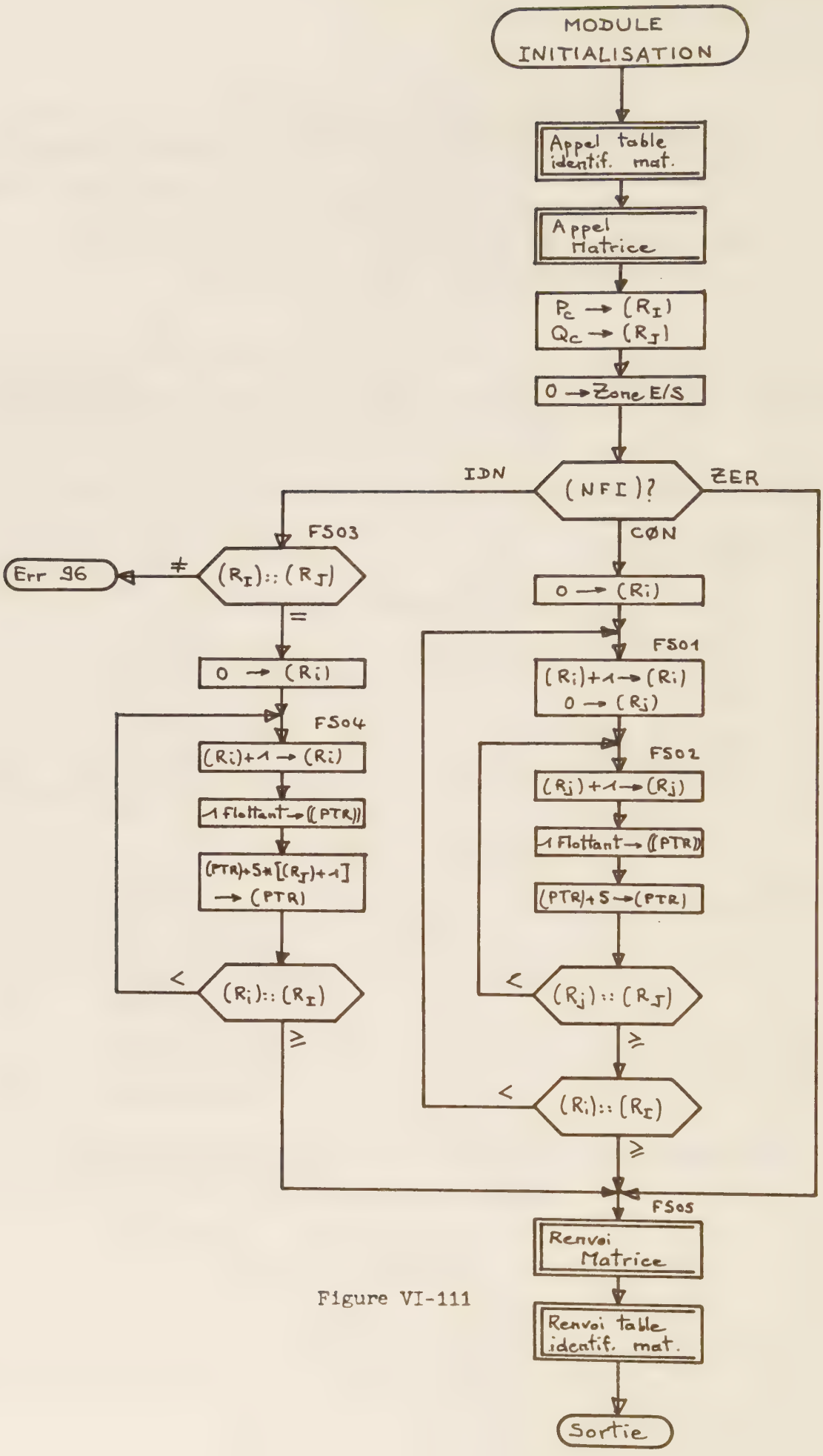


Figure VI-111

Le module initialisation (figure VI-111) commence par l'appel de la table des identificateurs matriciels grâce à laquelle on sait où aller chercher la matrice en tambour. Après prélèvement des dimensions, il y a remise à zéro de la zone d'entrée-sortie où se trouve la matrice, si la fonction est ZER, le travail est fini. Si la fonction est CØN, il y a mise à 1 en flottant dans tous les éléments de 5 octets sous contrôle des indices R_i et R_j. Si la fonction est IDN, on vérifie d'abord que les dimensions sont égales, avant de mettre 1 en flottant dans tous les éléments de la diagonale principale.

Le module s'achève de façon unique pour les 3 fonctions par le renvoi en tambour de la matrice puis de la table des identificateurs matriciels.

VI-22-5-2 - Fonctions matricielles

VI-22-5-2-1 - Définition - Forme générale

On appelle fonctions matricielles les fonctions TRN et INV qui effectuent respectivement la transposition et l'inversion d'une matrice.

La forme générale est la même pour les 2 types :

× × × × MAT nom de matrice = TRN (nom de matrice)
× × × × MAT nom de matrice = INV (nom de matrice)

Les dimensions des matrices figurant dans ces instructions doivent être compatibles :

Pour TRN $P_{c1} = Q_{c2}$ et $Q_{c1} = P_{c2}$

Pour INV $P_{c1} = Q_{c1} = P_{c2} = Q_{c2}$

Après transposition ou inversion de la matrice figurant à droite, il y a affectation du résultat à la matrice figurant à gauche.

VI-22-5-2-2 - Traitement

Etat de la pile T au 2è passage dans IDENTIFICATEUR DE TABLEAU pour l'exemple × × × × A = INV (B)

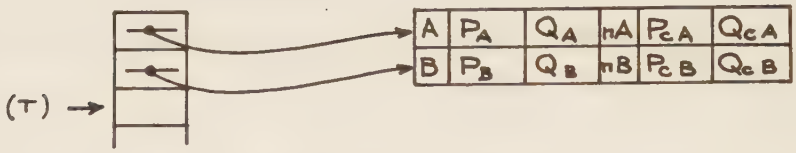


Figure VI-112

Le 1er passage a lieu par le branchement dû à AIG = 16 sur la détection du symbole d'affectation matricielle, avec ((P)-2) = <8>, code syntaxique du mot-clé MAT, et l'on passe par A775.

Au second passage, on dispose dans ((P)-2) du code syntaxique < 136 > obtenu dans OUVPAR et qui correspond à <fonction matricielle (>, on passe donc par A795 et on appelle le module FØNCTIØN MATRICIELLE.

Le module FONCTION MATRICIELLE (figure VI-115), commence par l'appel à la table des identificateurs matriciels, à l'aide du contenu de la pile T (figure VI-112), on peut prélever les dimensions courantes des 2 matrices concernées.

Le test sur NFM (Nom Fonction Matricielle) chargé dans TRN/INV (figure VI-114), permet de séparer le traitement de la transposition de celui de l'inversion. Dans les 2 cas, il y a vérification des compatibilités des dimensions.

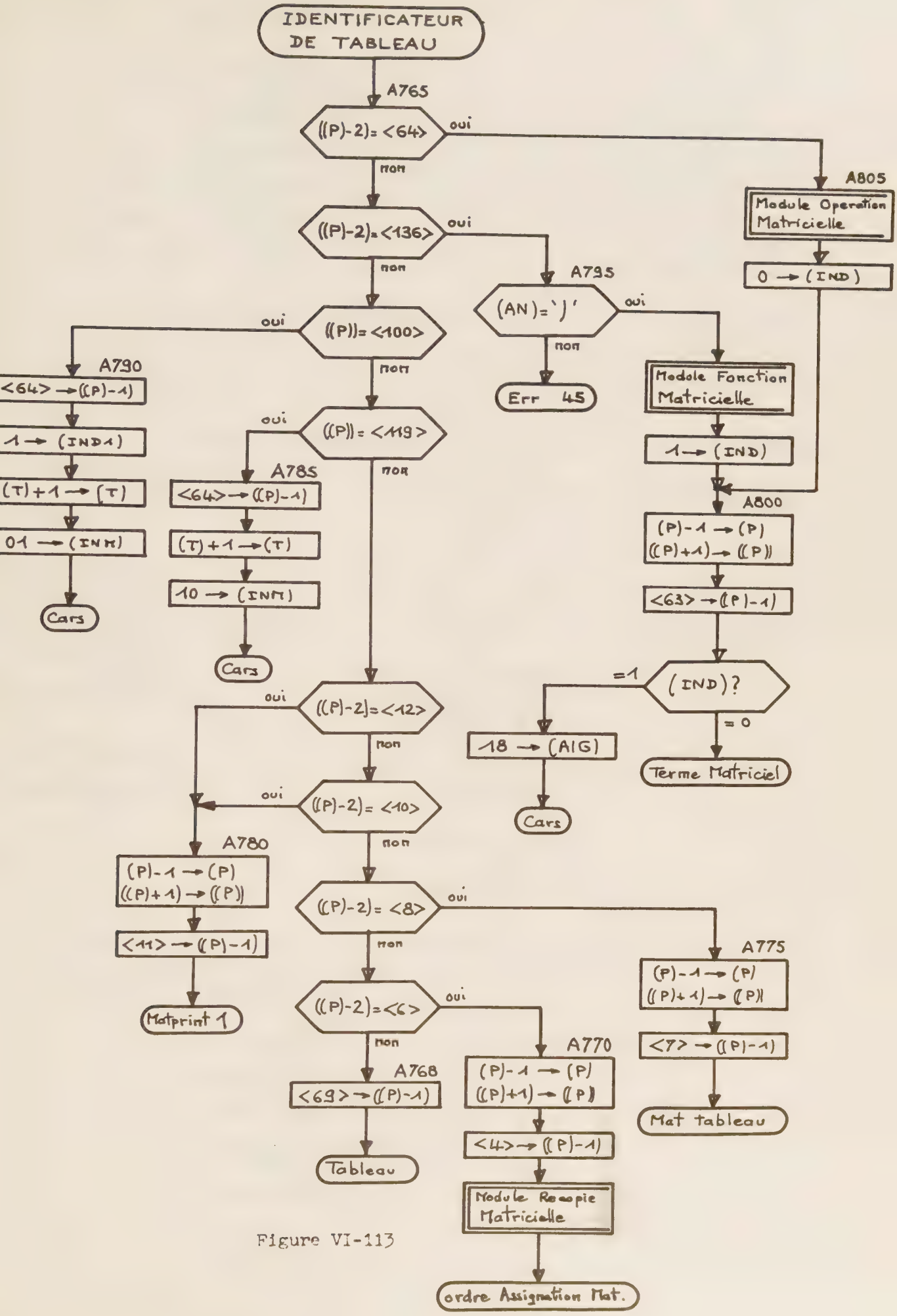


Figure VI-113

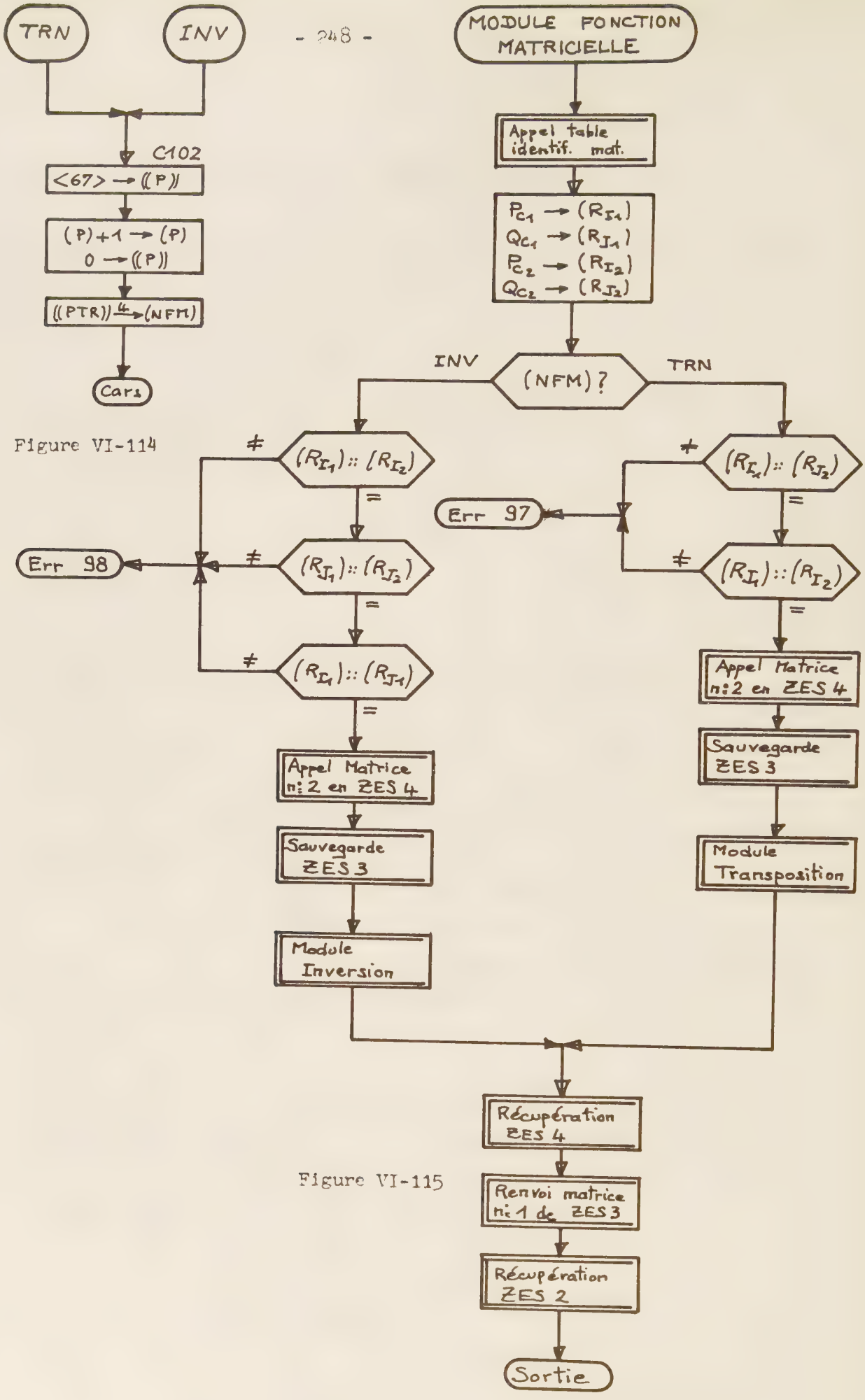


Figure VI-114

Figure VI-115

Le module INVERSION est trop important pour figurer ici, la méthode d'inversion utilisée est la méthode de GAUSS-JORDAN d'élimitation sur élément pivot, avec normalisation des pivots et échange de lignes si un pivot est nul. Cette méthode repose sur les propriétés des systèmes d'équations linéaires.

Le module TRANSPOSITION est plus trivial on prélève les éléments de la 2è matrice colonne par colonne et on les dépose en séquence dans la zone entrée-sortie affectée à la première matrice.

Remarquons que l'utilisation des zones entrée-sortie affectées au tambour est la même pour les 2 fonctions:

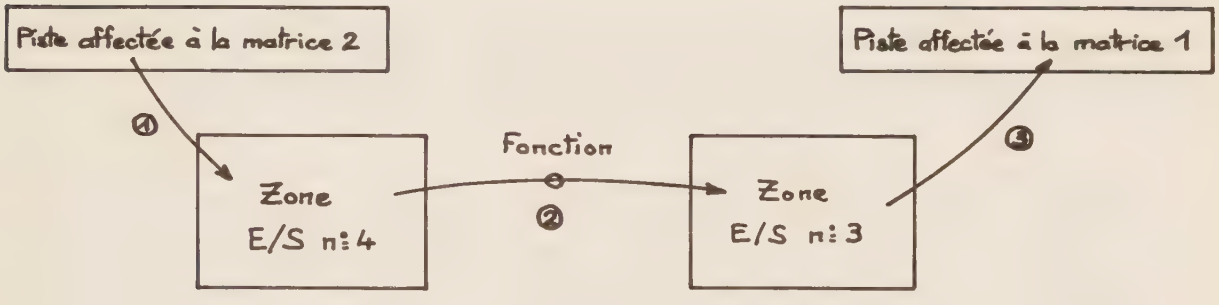


Figure VI-116

(Cette utilisation des zones entrée-sortie explique que l'on n'a pas employé ici la technique d'inversion sur place, qui permet normalement de gagner la moitié de l'encombrement dû à la technique normale, car au moment de l'inversion, l'encombrement général de la mémoire centrale ne pose pas de problème).

VI-22-5-3 - Opérations matricielles

VI-22-5-3-1 - Définitions - Formes générales

Les opérations matricielles utilisées ici, sont celles qui font partie du calcul matriciel courant, nous avons :

- La multiplication matricielle :

× × × × MAT nom de matrice = nom de matrice * nom de matrice.

où $P_{c1} = P_{c2}$ $\quad Q_{c1} = Q_{c3}$, $Q_{c2} = P_{c3}$

- L'addition/soustraction matricielle :

× × × × MAT nom de matrice = nom de matrice + nom de matrice
× × × × MAT nom de matrice = nom de matrice - nom de matrice

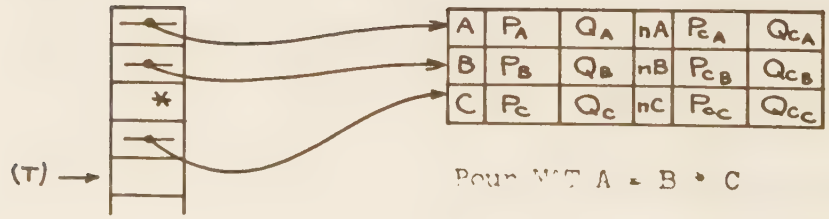
où $P_{c1} = P_{c2} = P_{c3}$, $Q_{c1} = Q_{c2} = Q_{c3}$

- La multiplication d'une matrice par un scalaire :

× × × × MAT nom de matrice = (expression) * nom de matrice

où $P_{c1} = P_{c2}$ et $Q_{c1} = Q_{c2}$.

VI-22-5-3-2 - Etat de la pile T dans les différents cas



Pour MAT A = B * C

Figure VI-117

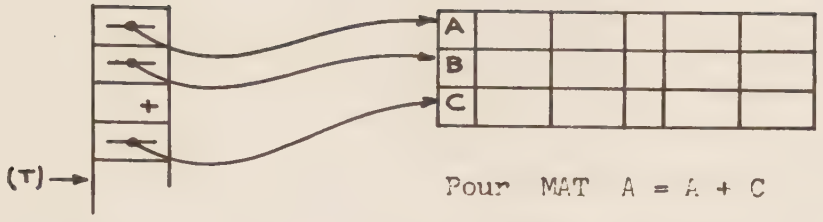


Figure VI-118

Pour MAT A = A + C

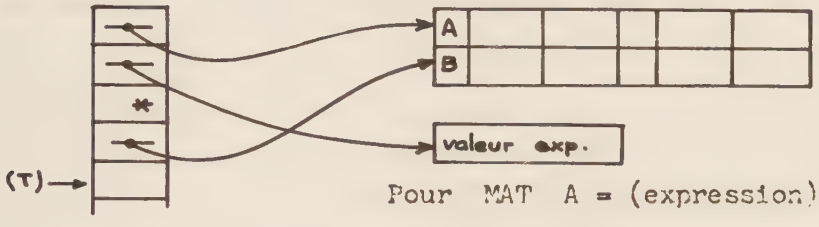


Figure VI-119

Pour MAT A = (expression) * B

VI-22-5-3-3 - Traitement

L'indicateur matriciel INM est positionné de façon à orienter le traitement dans le module ØPERATIØN MATRICIELLE (figure VI-120).

L'opération "multiplication par une constante" correspondant à INM = 0, INM étant positionné dans (EXPRESSION) (figure VI-34) en A540, puisque l'on a trouvé une expression entre parenthèses dans un contexte matriciel (IND7 = 1)

INM est mis à 1 dans IDENTIFICATEUR DE FONCTION (figure VI-113) à la détection de ((P)) =<100> , code correspondant au symbole de multiplication matricielle, de même INM est mis à 10 pour le symbole d'addition matricielle.

Dans chacun des traitements relatifs aux opérations matricielles, il y a vérification des compatibilités des dimensions, l'utilisation de la pile T permettant de prélever les dimensions courantes de chaque matrice dans les identificateurs matriciels convenables.

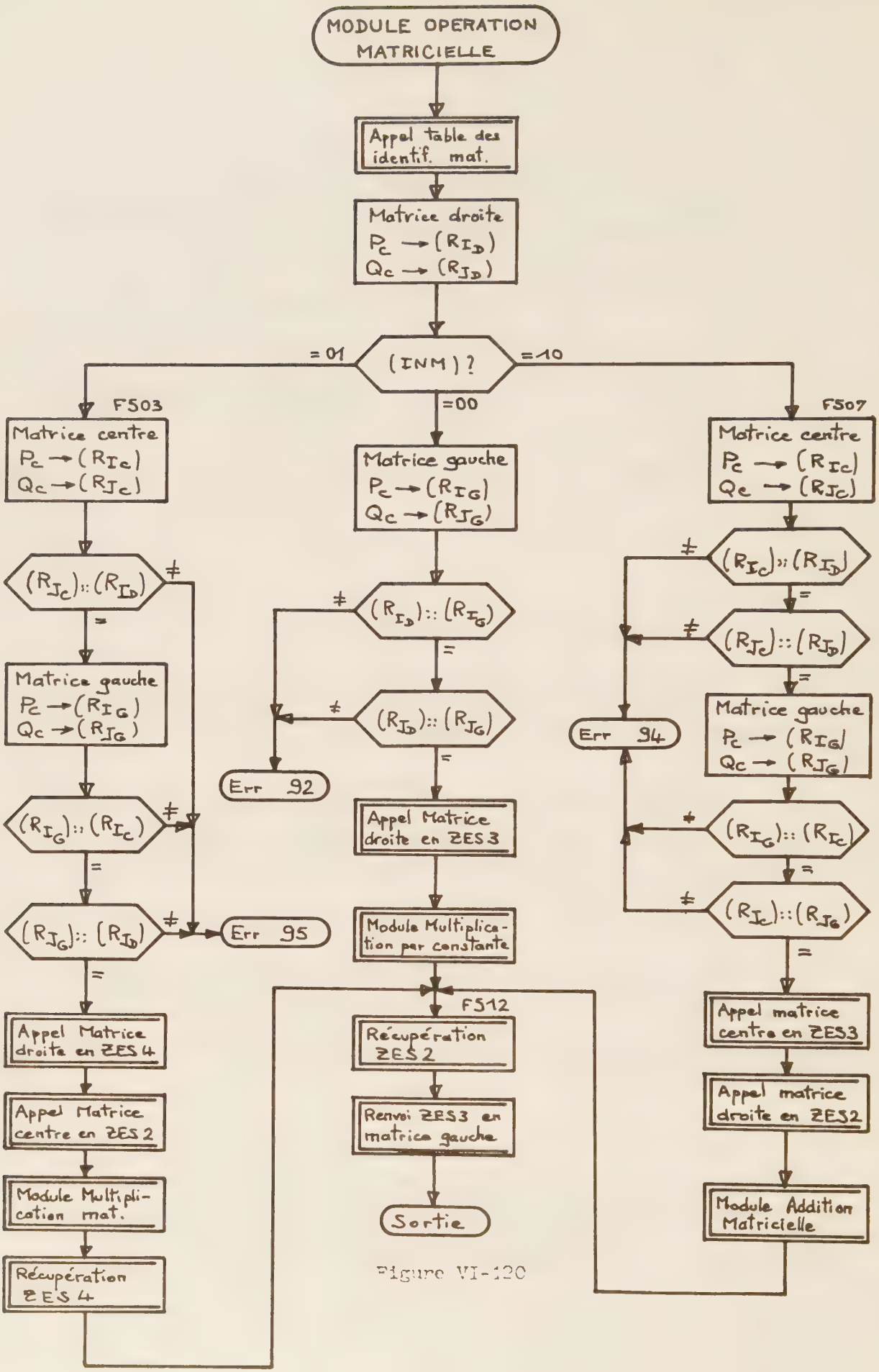


Figure VI-120

Le module "Multiplication matricielle" travaille avec 3 zones d'entrée-sortie, 2 recevant les matrices dont on effectue le produit, la 3ème recevant les éléments de la matrice produit, laquelle est affectée en F512 à la matrice de gauche.

Le module "Addition matricielle" travaille avec 2 zones, en effet, on peut utiliser l'une des zones pour contenir à la fois l'image d'une des matrices et le résultat de l'addition (ou de la soustraction).

Enfin le module "Multiplication par une constante" ne nécessite qu'une seule zone de travail, qui reçoit au départ l'image de la matrice dont on multiplie les éléments par la valeur de l'expression arithmétique, chaque nouvel élément prend la place de l'ancien dans la zone. On trouve dans une des cellules de la pile T, l'adresse de la cellule de 5 octets qui contient la valeur de l'expression

VI-22-5-4 - Recopie matricielle

VI-22-5-4-1 - Forme générale - But

× × × × MAT nom de matrice = nom de matrice

Le but de cette instruction est d'affecter aux éléments de la matrice de gauche la valeur des éléments de la matrice de droite.

Cette instruction permet d'éviter l'emploi de l'instruction "multiplication par une constante" ; la constante employée étant l'unité. Par exemple l'instruction MAT A = (1) * B est remplacée par MAT A = B.

VI-22-5-4-2 - Traitement

Comme auparavant, la pile T contient les adresses des identificateurs matriciels :

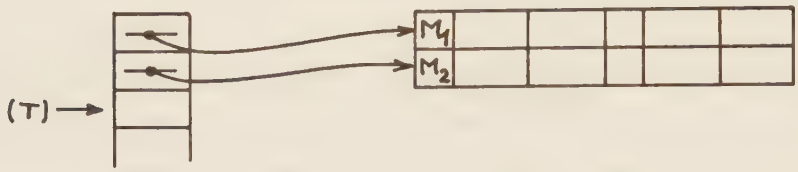


Figure VI-121

Au second passage dans IDENTIFICATEUR DE TABLEAU (figure VI-113) la présence dans ((P)-2) du code < 6 > (qui correspond à l'analyse de "MAT A =") permet de déclencher la recopie matricielle.

Le module (figure VI-122), comporte après prélèvement des dimensions une séquence de vérifications de compatibilité des dimensions. La recopie est triviale, il y a lecture de la matrice droite et attribution de la zone entrée-sortie 3 ainsi garnie, à la matrice de gauche.

VI-23 - INTERRUPTIONS NORMALES ET ERREURS

VI-23-1 - Routine de listage

Lors d'une interruption, il peut être intéressant de sortir à l'imprimante la liste du programme qui vient d'être exécuté, le compilateur valide pour cela le clavier numérique sur lequel on frappe la

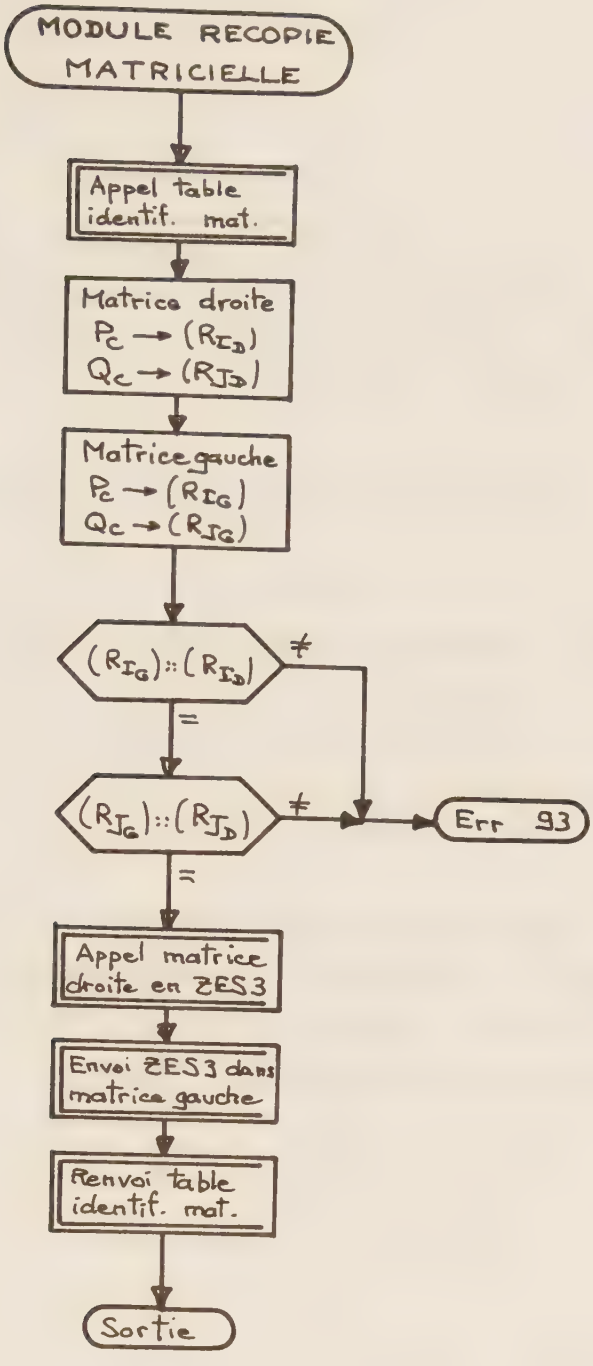


Figure VI-122

réponse : 0 pour non, 1 pour oui.

Si la réponse est positive, on fait appel à la routine de listage. Quand la liste est obtenue, le compilateur perd la commande et celle-ci est prise en main par le moniteur-superviseur, c'est le mode de fonctionnement normal du système employé.

Si la réponse est 0, on fait appel directement au moniteur-superviseur.

VI-23-2 - Signalisation des erreurs d'interprétation

En cas d'interruption anormale du programme, il est important de signaler à l'utilisateur le numéro de l'erreur qui a provoqué l'interruption, ainsi que le numéro de l'instruction où l'erreur est apparue.

Les erreurs d'interprétation portent des numéros compris entre 68 et 99, sachant que par ailleurs les étiquettes des instructions sont numérotées à l'aide de 4 chiffres, on peut utiliser l'écran de visualisation à 6 chiffres pour signaler les erreurs.

Figure VI-123

numéro de l'erreur — numéro de l'instruction

Le système reste alors en attente sur cette visualisation, l'intervention de l'utilisateur a pour effet de débloquer le compilateur et de passer à la phrase de listage évoquée ci-dessus.

On trouvera en annexe la liste des erreurs d'interprétation avec leurs diagnostics.

CHAPITRE VII

PRATIQUE DU BASIC SUR GE 55

VII-1 - INTRODUCTION DU COMPILATEUR EN MACHINE

Nous avons vu que la compilation comporte deux phases distinctes : l'analyse et l'interprétation, qui correspondent à deux programmes séparés : l'analyseur dont il existe une version ANALY sans arbre d'analyse, et une version ANARB avec sortie de l'arbre d'analyse syntaxique pour chaque instruction, l'interpréteur qui porte le nom BASIC.

Ces programmes, dont les mises en oeuvre sont identiques, demandent à être assemblés ; pour cela, il y a introduction de l'assembleur en tambour (une seule fois, tant qu'on veut faire assembler des programmes), puis au lecteur de cartes, d'une séquence "Appel Assembleur" suivie du programme à assembler.

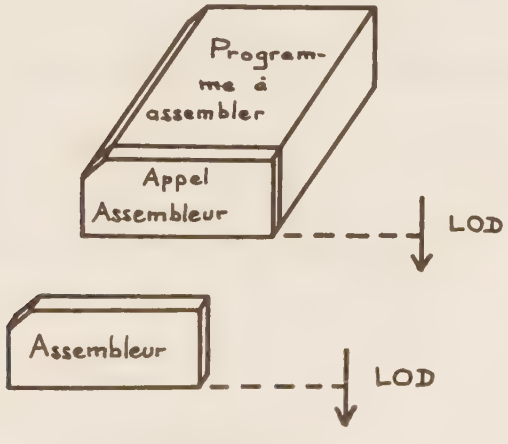


Figure VII-1

L'introduction des routines "Assembleur" et "Appel Assembleur" s'effectue par appel de la fonction chargement software LOD qui est câblée (appui sur le bouton LOD).

Le programme à assembler entre en machine sous le contrôle de l'assembleur.

Après assemblage, on dispose en tambour, du programme sous forme exécutable. Pendant la mise au point, on appelle alors le programme de la même façon qu'en exploitation (voir ci-dessous).

Pour obtenir un produit utilisable en exploitation, il convient de sortir un "binaire" chargeable à volonté. Ce "binaire" est obtenu au perforateur de cartes sous contrôle d'une routine "Vidage Tambour", qui, grâce à la carte de commande, va rechercher en tambour le programme et le perfore sous forme condensée (dans la mesure où l'on perfore sur un maximum de colonnes).

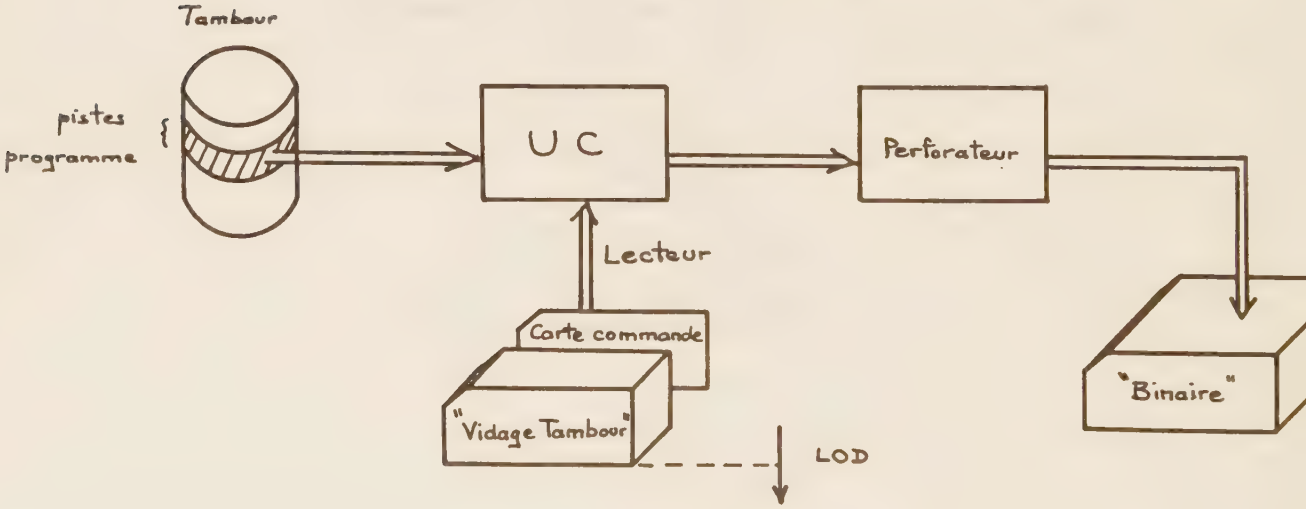


Figure VII-2

En exploitation, il suffit de charger le programme binaire à l'aide du "Chargeur-Tambour", dans des pistes consécutives choisies par le moniteur-superviseur en fonction de l'encombrement du tambour.

La carte de commande est toujours la même pour un programme donné, elle comporte outre son code-carte particulier, le nom du programme (ici, ANARB, ANALY ou BASIC).

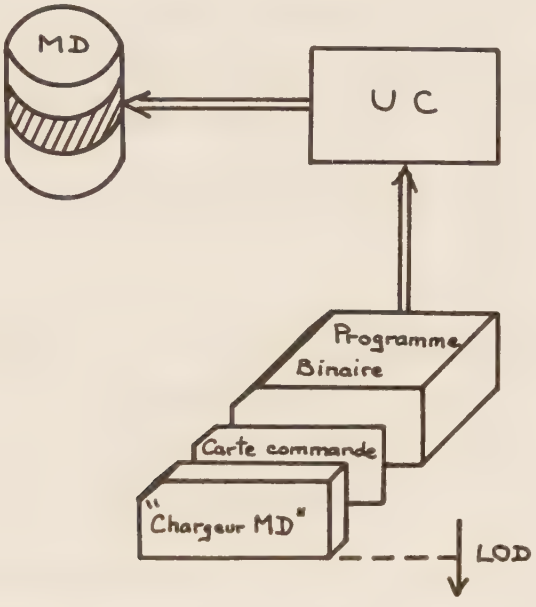


Figure VII-3

Une fois le programme chargé en tambour, on peut l'utiliser autant de fois que l'on veut, la procédure à suivre consiste à introduire en mémoire centrale l'"Appel Moniteur" suivi de la carte de commande.

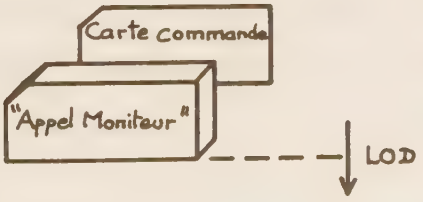


Figure VII-4

Le moniteur-superviseur va chercher en tambour le premier segment du programme et l'introduit en mémoire centrale, il y a alors transfert de commande et le programme prend la main.

VII-2 - ANALYSEUR SYNTAXIQUE

L'analyseur syntaxique est introduit en machine de la façon exposée ci-dessus, l'analyseur en exécution ne demande pas l'utilisation du lecteur de cartes, l'entrée des instructions BASIC se faisant au clavier alphanumérique.

Le synoptique du fonctionnement est celui qui est dessiné à la figure VII-5.

On dispose de deux ordres de commande, l'un LIST permet d'obtenir la liste des instructions BASIC déjà entrées et triées, ceci à tout moment, l'autre PUNCH commande la perforation du programme BASIC pour obtenir un fichier exploitable par l'interpréteur.

Remarquons que selon la version de l'analyseur, l'impression de l'instruction est suivie (ANARB) ou non (ANALY) de l'impression des codes syntaxiques qui permettent de retrouver l'arbre d'analyse syntaxique. On trouvera en annexe D des exemples qui éclaireront ce point. Dans le cas d'une erreur syntaxique, l'impression de la table ARB s'arrête à l'élément qui a permis de détecter l'erreur.

VII-3 - INTERPRETEUR

En interprétation, la carte de commande qui porte le nom du programme, BASIC, est suivie du fichier extrait du perforateur dans la phase analyse.

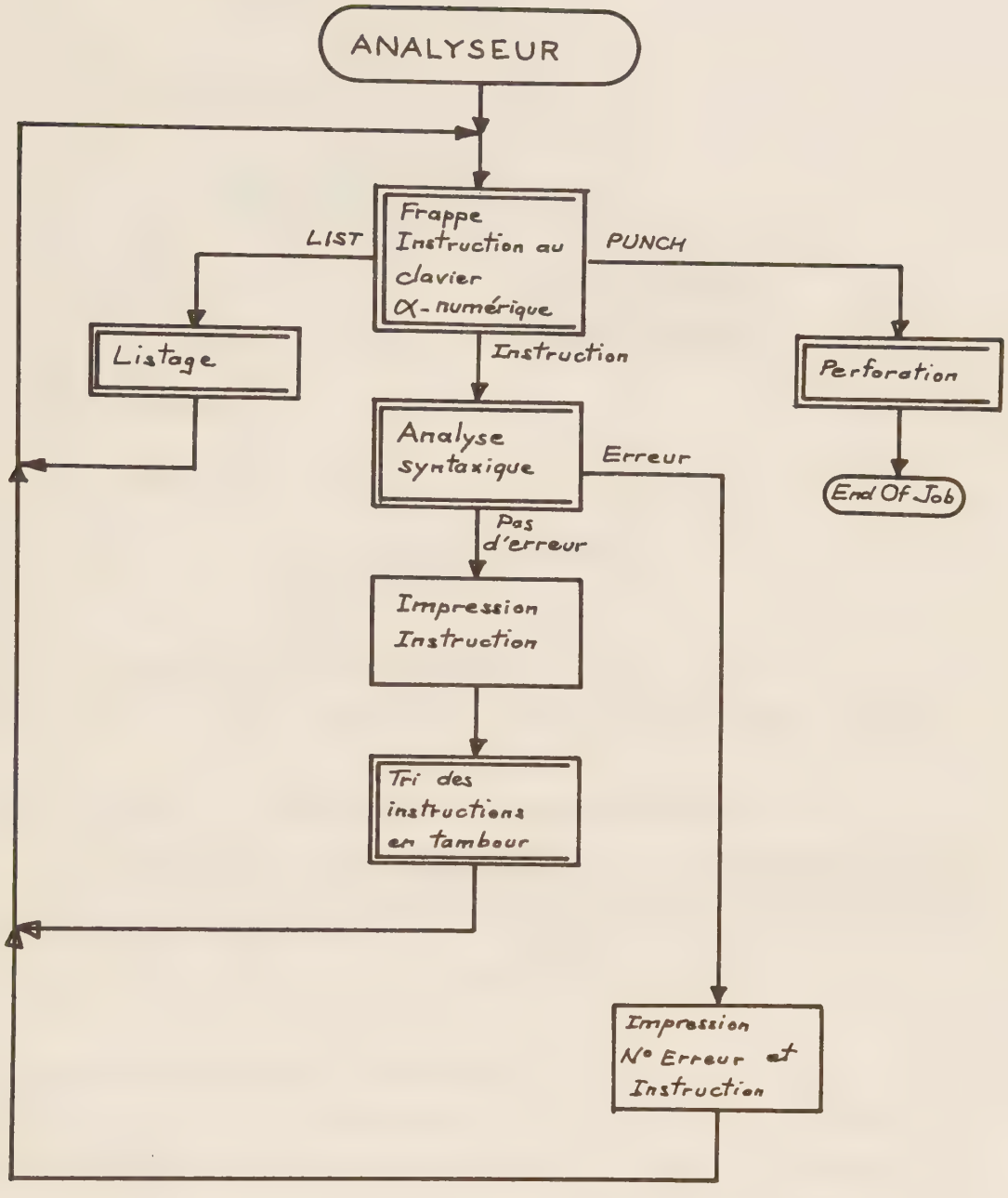

ANALYSEUR
Frappe Instruction au clavier α-numérique
LIST
PUNCH
Listage
Instruction
Perforation
Analyse syntaxique
Erreur
End Of Job
Pas d'erreur
Impression Instruction
Tri des instructions en tambour
Impression N° Erreur et Instruction

Figure VII-5

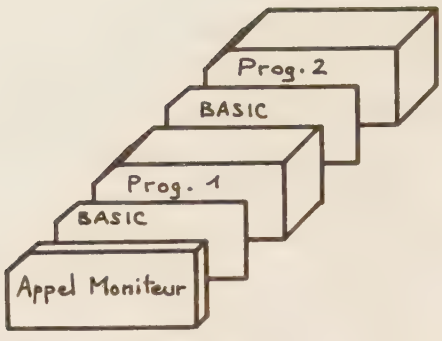


Figure VII-6

En fait, chaque programme BASIC est suivi d'une carte fin de fichier comportant /* pour code-carte.

On peut répéter le jeu carte de commande + fichier BASIC autant de fois que l'on veut. En effet, chaque programme BASIC entre en machine sous contrôle de l'interpréteur. En fin d'exécution, le moniteur reprend la commande, et si l'on présente à nouveau une carte d'appel de l'interpréteur BASIC, le processus recommence.

On peut donner un synoptique de l'interpréteur (figure VII-7). La lecture des instructions du programme BASIC se fait avec recherche des mots-clés FØR, NEXT et DATA en vue de l'analyse complémentaire (décrite au § VI-1). Toutes les autres instructions sont enregistrées par blocs de 9 en tambour. La détection du code fin de fichier permet l'achèvement de l'analyse complémentaire et le lancement de l'exécution du programme, les ordres d'entrée/sortie mettent en jeu soit le clavier alphanumérique, soit l'imprimante. Après visualisation d'une erreur, l'intervention de l'utilisateur a pour effet de revenir à la séquence d'interruption normale, en vue de sortir la liste du programme ou de repasser la commande au moniteur.

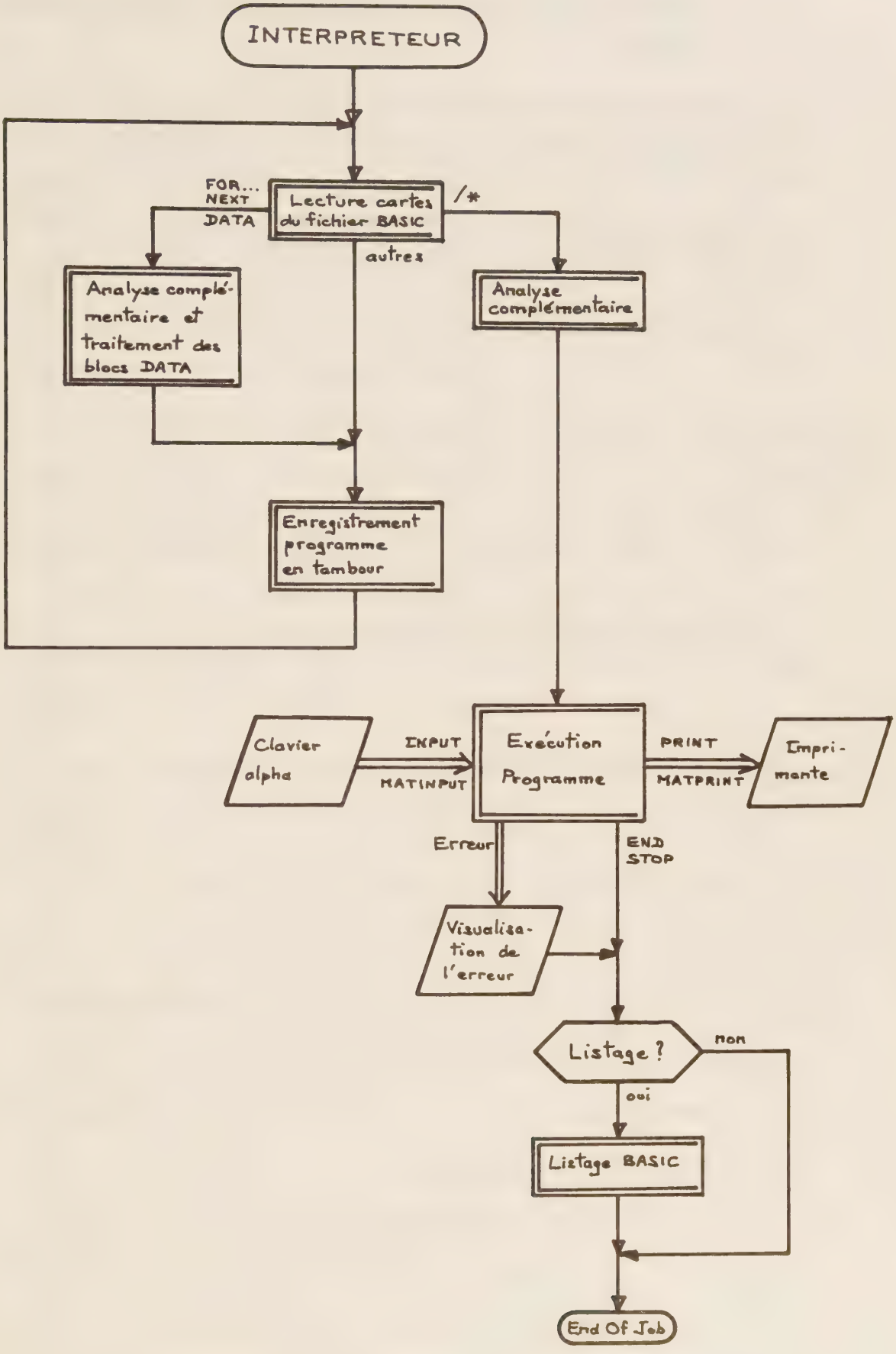

INTERPRETEUR
FOR...
NEXT
DATA
Lecture cartes du fichier BASIC
/*
autres
Analyse complémentaire et traitement des blocs DATA
Analyse complémentaire
Enregistrement programme en tambour
Clavier alpha
INPUT
MATINPUT
Exécution Programme
PRINT
MATPRINT
Imprimante
Erreur
END
STOP
Visualisation de l'erreur
Listage ?
non
oui
Listage BASIC
End Of Job

Figure VII-7

VII-4 - ORGANISATION DE LA MEMOIRE CENTRALE ET DU TAMBOUR EN INTERPRETATION

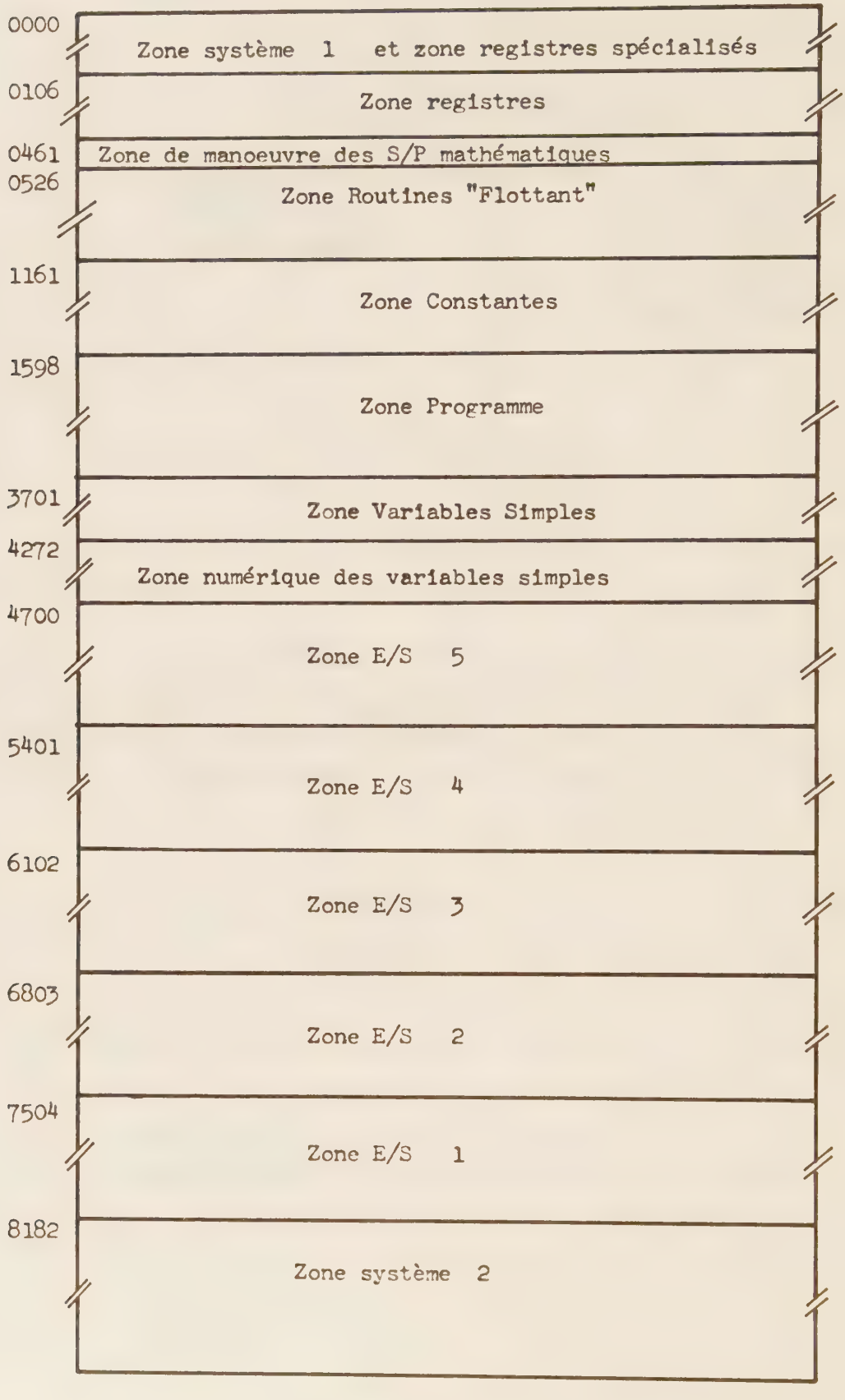


Figure VII-8

La zone "Flottant" est implantée de façon fixe depuis l'octet 526 jusqu'à l'octet 1160, ainsi que la zone de manoeuvre des sous-programmes mathématiques depuis l'octet 461 jusqu'à 525. La zone registres est alors limitée entre 106 (le registre 1 est réservé au système) et 460, soit depuis le registre 2 jusqu'au registre 72 inclus (certains registres de base 6, 8, 9, 10, 11, 12, sont garnis par le moniteur et sont réservés aux zones entrée/sortie du système qui figurent dans la zone système 2).

Derrière la zone "Flottant", on trouve la zone des constantes entre 1161 et 1597, elle contient tous les codes syntaxiques supérieurs à <99> , la position d'analyse, la position d'aiguillage, les indicateurs, les adresses initiales et finales des piles, etc...

La zone programme (1898 à 3700) reçoit les segments du programme, ces segments sont constitués de 3 pistes consécutives du tambour (soit 2100 octets).

La zone des variables simples (3701 à 4272) reçoit (voir § VI-3) les adresses des cellules de 5 octets de la zone numérique suivante (4272 à 4699).

On trouve ensuite les zones entrée/sortie de communication avec le tambour. La zone E/S 1 a 675 octets (+ 1 drapeau) de long et représente la longueur d'un bloc BASIC (9 fois 75 octets), cette zone reçoit donc le programme BASIC bloc par bloc, elle reçoit également au moment opportun, la liste enchaînée des mots-clés, et la pile des fonctions.

La zone E/S 2 reçoit la piste des identificateurs matriciels, la table des renseignements sur les DATA, la piste des fonctions, la piste des boucles SP.

La zone E/S 3 reçoit des matrices.

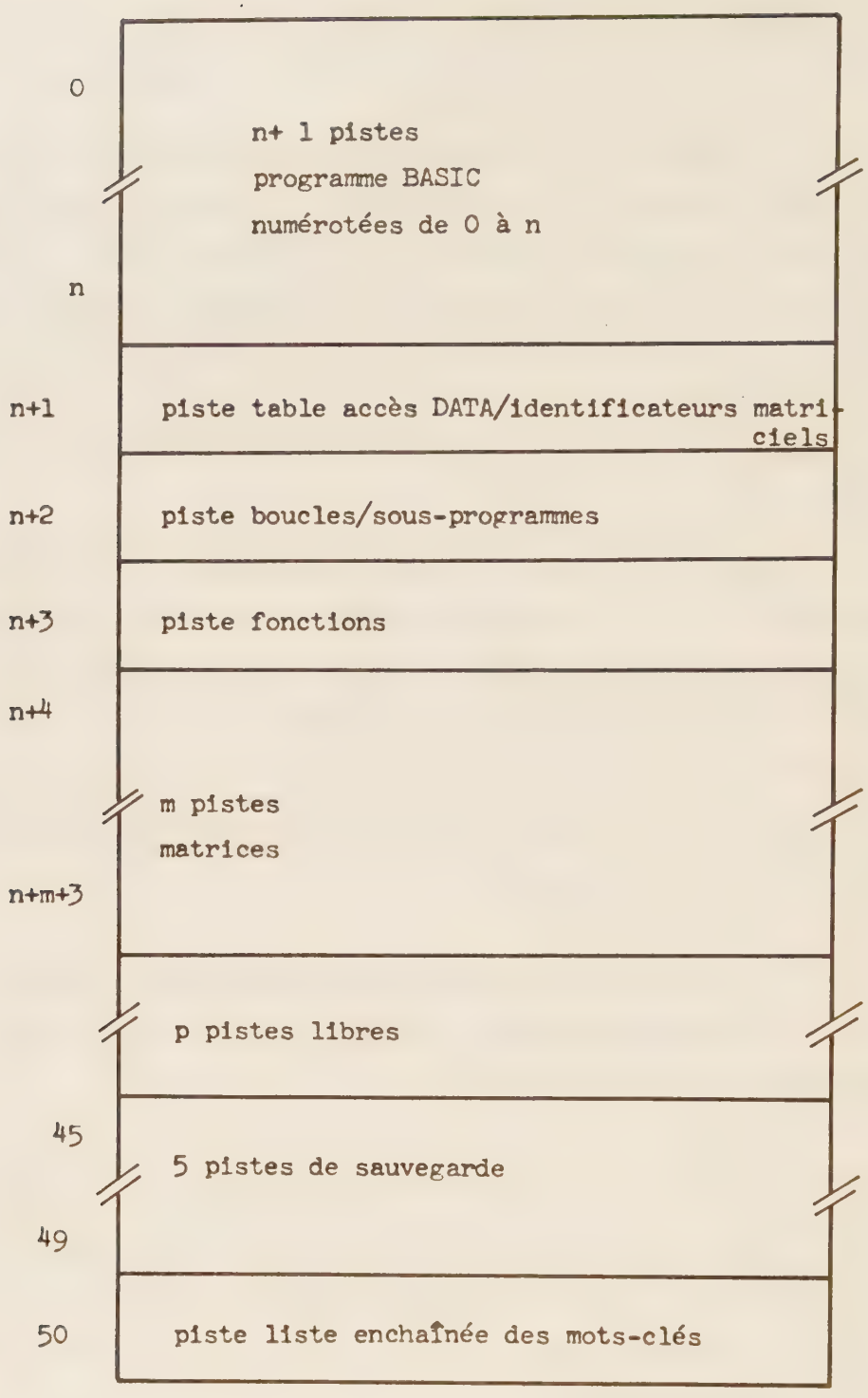

0
n+ 1 pistes
programme BASIC
numérotées de 0 à n
n
n+1
piste table accès DATA/identificateurs matriciels
n+2
piste boucles/sous-programmes
n+3
piste fonctions
n+4
m pistes
matrices
n+m+3
p pistes libres
45
5 pistes de sauvegarde
49
50
piste liste enchaînée des mots-clés

Figure VII-9

La zone E/S 4 reçoit des matrices (dans le cas de fonctions et d'opérations matricielles), elle contient le plus souvent la pile d'analyse.

Enfin, la zone E/S 5 ne reçoit jamais de pistes, elle contient la pile d'interprétation et la pile auxiliaire.

Le programme BASIC est enregistré dans n+1 pistes, n variant avec la longueur du programme, les 3 pistes qui suivent, reçoivent les informations nécessaires au traitement des DATA, des identificateurs de tableaux, des boucles, des sous-programmes et des fonctions.

On trouve ensuite m pistes matrices, m étant le nombre total de matrices figurant dans les ordres DIM.

p pistes restent libres.

Les 5 zones d'entrée/sortie sont sauvegardées (en général) dans les pistes 45 à 49, avant de recevoir le contenu d'autres pistes.

La 50ème piste contient par constantes tambour (chargées par le système) la liste enchaînée des mots-clés.

VII-5 - CONTRAINTES DE L'INTERPRETEUR

Pour qu'il n'y ait pas de débordement des pistes matrices sur les pistes de sauvegarde, on doit avoir :

$$n + m + 3 \leqslant 44$$

soit $n + m \leqslant 41$

Si le nombre d'instructions du programme est I, le nombre de blocs est alors $n+1 = \text{Ent}(\frac{I+8}{9})$

D'où : $\text{Ent}((I+8)/9) + m \leq 42$

Exemple :

Si le nombre de matrices définies est 10, on peut avoir un programme de 288 instructions.

On peut considérer sur ce point que le compilateur ainsi conçu n'est pas très restrictif.

Le nombre de matrices réservées ne doit pas dépasser 13.

Le nombre de fonctions internes ne doit pas dépasser 11.

Le nombre de sous-programmes encastrés ne doit pas dépasser 20.

Le nombre de boucles emboîtées ne doit pas dépasser 10.

Ces limites ont été choisies en fonction de certaines particularités de programmation relatives à la dimension des tables employées, selon les types des cellules associées aux ordres considérés.

VII-6 - TEMPS D'EXECUTION

VII-6-1 - Temps relatifs à l'arithmétique flottante

VII-6-1-1 - Durée d'exécution des opérations élémentaires

Addition	110 à 133 ms
Soustraction	115 à 137 ms
Multiplication	163 à 185 ms
Division	378 à 400 ms
Comparaison	132 à 154 ms
Exponentiation	$\simeq 5{,}4$ s $\quad p = 2.10^{-6}$

VII-6-1-2 - Durée Moyenne d'exécution des fonctions prédéfinies

ABS	275 ms	
ATN	2,8 s	$p = 2.10^{-6}$
COS	2,5 s	$p = 2.10^{-6}$
EXP	2,2 s	$p = 2.10^{-6}$
LOG	3,2 s	$p = 2.10^{-6}$
SIN	2,5 s	$p = 2.10^{-6}$
SQR	2,1 s	$p = 10^{-7}$
TAH	3,2 s	$p = 5.10^{-6}$

VII-6-1-3 - Conversions

La conversion nécessaire en entrée, soit à la lecture dans un DATA, soit à la frappe d'une donnée numérique, requiert environ 2 s.

La conversion en sortie impression est très variable en durée, mais l'ordre de grandeur reste le même, environ 2 s par quantité à imprimer.

VII-6-1-4 - Macro-Instructions Système

La programmation en système tambour est facilitée par l'emploi de macro-instructions dont le développement au niveau de l'assemblage se résume à la génération d'une séquence d'appel à une routine spécialisée du superviseur.

Par exemple, la demande de lecture (RDR) ou d'écriture (WDR) d'une piste en tambour est prise en charge par une routine qui vérifie entre autres l'appartenance de la piste au module en exécution et qui déclenche réellement l'opération d'entrée-sortie. L'étude de cette routine nous apprend que la durée d'exécution de ces macro-instructions est de l'ordre de 115 ms.

La macro-instruction CSG (Call Segment) est générée par l'assembleur en vue d'assurer le fonctionnement en recouvrement (overlay) du compilateur, la prise en charge demande environ 200 ms avec des segments constitués de 3 pistes, du même ordre que l'appel et le retour d'un sous-programme (qui ne fait pas partie du même segment que la séquence d'appel).

VII-6-1-5 - Etude de certains mécanismes de l'interpréteur

En additionnant les temps des instructions qui réalisent un traitement donné, on trouve une bonne appréciation de la durée moyenne de ce traitement.

Ainsi, il faut environ 100 ms pour prendre en charge la routine CARS, et accéder à la bonne adresse dans la table de transfert des caractères terminaux. Sachant que la séquence de transfert est en fait 90 % du temps, une séquence avec changement de segment, c'est au total 300 ms qui sont nécessaires à la prise en charge du caractère suivant (mot-clé et libellés mis à part) jusqu'à l'entrée dans la routine de traitement de ce caractère.

Pour un mot-clé, la reconnaissance de la première lettre correspond à une durée de 300 ms (voir ci-dessus), il faut ensuite appeler la liste enchaînée après avoir reconnu que le caractère suivant est aussi une lettre, soit 20 ms pour la lettre et 2 fois 115 ms pour RDR et WDR (mise en garde Zone E/S 1), il faut compter en moyenne 15 ms par caractère pour se déplacer dans la liste (avec identité ou non des caractères), il est assez difficile de connaître le temps d'accès moyen au code du mot-clé, mais pour se donner une idée, avec 15 lettres différentes pour première lettre d'un mot-clé, il faut en moyenne 7 x 15 $\simeq$ 100 ms pour retrouver le début du mot-clé dans la liste et, mettons 100 ms pour toutes les autres lettres.

Enfin, on traite le symbole @ de fin de mot-clé (50 ms), puis on récupère la zone originelle (115 ms) et on se branche à la routine de traitement du mot-clé (200 ms).

Au total, il faut 1,115 s par mot-clé en moyenne.

Ces deux exemples montrent, d'une part que la faible capacité de la mémoire se paie cher, d'autre part que le fonctionnement en recouvrement est cause d'une perte de temps importante surtout quand la routine traitée dans le segment appelé est courte et entraîne rapidement un rappel du segment appelant.

Tout ceci explique finalement des performances médiocres qui sont pour la plupart dues à la machine. Il y a des causes de ralentissement dues à la nature du compilateur, il est bien clair que la réinterprétation multiple des instructions inclues dans une boucle ne peut que faire perdre du temps.

A titre d'exemple, nous avons effectué des passages de programmes identiques en BASIC et en MINIFORTRAN qui est un compilateur génératif. Ce dernier est plus rapide dès qu'il y a des boucles (de l'ordre de 3 fois plus rapide), par contre il est plus lent s'il n'y en a pas (de l'ordre de 2 fois), dans des conditions identiques, c'est-à-dire avec présentation du même programme source au compilateur.

Ces exemples semblent être défavorables à notre compilateur, mais on verra dans la conclusion, comment on peut améliorer les performances de façon très sensible.

CHAPITRE VIII

EXEMPLES DE PROGRAMMES BASIC EXECUTES SUR GE 55

VIII-1 - LISTE DES EXEMPLES

VIII-1-1 - Résolution de l'équation du second degré

VIII-1-2 - Simulation de l'opération logique ET

VIII-1-3 - Trigonométrie - Calcul de la longueur du 3ème côté d'un triangle connaissant les deux autres et l'angle qu'ils forment

VIII-1-4 - Calcul des logarithmes en base 2, e et 10

VIII-1-5 - Génération d'une table de Sinus/Cosinus

VIII-1-6 - Génération d'une table de racines carrées

VIII-1-7 - Calcul des racines carrée, cubique et quatrième

VIII-1-8 - Modification d'un élément de matrice

VIII-1-9 - Initialisation d'un sous-tableau

VIII-1-10- Affectation matricielle

VIII-1-11- Addition - Soustraction matricielle - Multiplication par une constante

VIII-1-12- Multiplication matricielle

VIII-1-13- Opérations matricielles

VIII-1-14- Entrée directe d'une matrice

VIII-1-15- Initialisations matricielles

VIII-1-16- Opération matricielle

VIII-1-17- Lecture de matrices dans les DATA

VIII-1-18- Multiplication matricielle dans les 2 sens

VIII-1-19- Elévation d'une matrice à une puissance

VIII-1-20- Multiplication d'une matrice par un vecteur-colonne

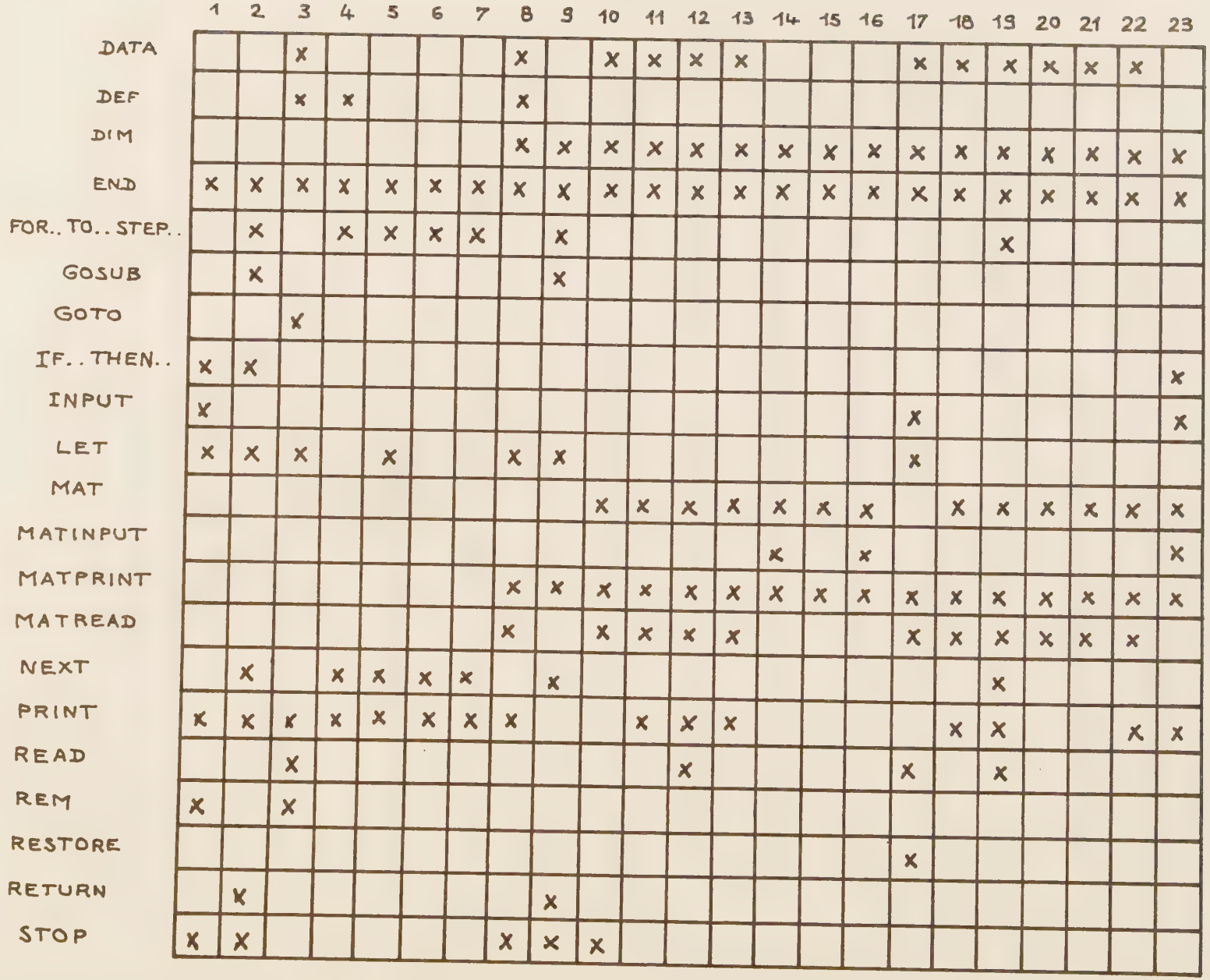
VIII-1-21- Transposition de matrice

VIII-1-22- Inversion d'une matrice de Hilbert de rang 3

VIII-1-23- Inversions successives - Recherche des erreurs d'arrondi.

VIII-2 - TABLEAU DE DISTRIBUTION DES INSTRUCTIONS

Ce tableau montre que toutes les instructions appartenant au BASIC ont été employées. Il permet, en outre, de retrouver les exemples d'utilisation d'un ordre donné.


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
DATA
DEF
DIM
END
FOR..TO..STEP..
GOSUB
GOTO
IF..THEN..
INPUT
LET
MAT
MATINPUT
MATPRINT
MATREAD
NEXT
PRINT
READ
REM
RESTORE
RETURN
STOP

VIII-3 - EXEMPLES

```
A*X↑2+B*X+C=0

     COEFFICIENTS                A= 3     B= 5.5      C= 8

RACINES COMPLEXES
     X1=-0,9166667      +J 1,35144
     X2=-0,9166667      -J 1,35144
```

LIST

```
0005   REM  RESOLUTION DE L'EQUATION DU SECOND DEGRE
0006   PRINT "A*X↑2+B*X+C=0"
0007   PRINT
0008   INPUT A,B,C
0009   PRINT "     COEFFICIENTS     ","A=";A;"B=";B;"C=";C
0010   LET D=B*B-4*A*C
0011   PRINT
0015   IF D<0 THEN 35
0026   PRINT "RACINES REELLES"
0027   PRINT "     X1=";(-B+SQR(D))/(2*A)
0028   PRINT "     X2=";(-B-SQR(D))/(2*A)
0030   STOP
0035   LET X1=-B/(2*A)
0040   LET Y1=SQR(-D)/(2*A)
0045   LET X2=X1
0051   PRINT "RACINES COMPLEXES"
0052   PRINT "     X1=";X1;"+J";Y1
0053   PRINT "     X2=";X1;"-J";Y1
9999   END
```

VIII-3-1 - Résolution de l'équation du second degré

```
 A              B              A.AND.B

FAUX           FAUX           FAUX
FAUX           VRAI           FAUX
VRAI           FAUX           FAUX
VRAI           VRAI           VRAI
```

```
                                LIST

0010    PRINT "  A","  B","A.AND.B"
0020    PRINT
0030    FOR A=0 TO 1
0040       FOR B=0 TO 1
0050       LET X=A
0060       GOSUB 500
0070       LET X=B
0080       GOSUB 500
0090    LET X=A*B
0100    GOSUB 500
0110       PRINT
0120       NEXT B
0130    NEXT A
0140    STOP
0500    IF X=0 THEN 530
0510    PRINT"VRAI",
0520    RETURN
0530    PRINT "FAUX",
0540    RETURN
9999    END
```

VIII-3-2 - Simulation de l'opération logique ET

```
COTE          ANGLE         COTE          3EME COTE
 10            60            20            17,32051
 5             90            4             5
 10            120           10            17,3205
 5             60            5             4,999999
```

```
LIST

0100    REM TRIGONOMETRIE
0110    LET P=3,14159265
0120    DEF FND(X)=X*P/180
0130    DEF FNC(X)=COS(FND(X))
0140    PRINT "COTE","ANGLE","COTE","3EME COTE"
0150    READ X,A,Y
0160    LET Z=SQR(X*X+Y*Y-2*X*Y*FNC(A))
0170    PRINT X,A,Y,Z
0180    GOTO 150
0190    DATA 10,60,20
0200    DATA 3,90,4
0210    DATA 10,120,10
0220    DATA 5,60,5
9999    END
```

VIII-3-5 - Trigonométrie

```
X             LOG BASE 2    LOG BASE E    LOG BASE 10

2              1             0,6931469     0,3010299
4              2,000002      1,386295      0,6020603
6              2,584964      1,79176       0,7781515
8              3,000002      2,079442      0,9030902
10             3,321929      2,302585      1
```

```
LIST

0010    DEF FNB(X)=LOG(X)/LOG(2)
0020    DEF FND(X)=LOG(X)/LOG(10)
0025    PRINT" X","LOG BASE 2 ","LOG BASE E","LOG BASE 10"
0028    PRINT
0030    FOR I=2 TO 10 STEP 2
0040       PRINT I,FNB(I),LOG(I),FND(I)
0050    NEXT I
9999    END
```

VIII-3-4 - Calcul des logarithmes en base 2, e et 10

ARC	SINUS	COSINUS	ARC
0	0	0,9999999	90
1	1,745239E-02	0,9998476	89
2	3,489948E-02	0,9993907	88
3	5,233593E-02	0,9986294	87
4	6,975644E-02	0,997564	86
5	8,715572E-02	0,9961946	85
6	0,1045284	0,9945218	84
7	0,1218694	0,9925461	83
8	0,139173	0,990268	82
9	0,1564345	0,9876883	81
10	0,1736481	0,9848078	80
11	0,1908089	0,9816273	79
12	0,2079117	0,9781475	78
13	0,2249509	0,9743699	77
14	0,2419218	0,9702957	76
15	0,2588189	0,9659259	75
16	0,2756372	0,9612615	74
17	0,2923716	0,9563048	73
18	0,3090169	0,9510566	72
19	0,325568	0,9455185	71
20	0,34202	0,9396926	70
21	0,3583678	0,9335804	69
22	0,3746065	0,9271836	68
23	0,390731	0,9205048	67
24	0,4067365	0,9135455	66
25	0,4226181	0,9063079	65
26	0,438371	0,8987939	64
27	0,4539903	0,8910065	63
28	0,4694714	0,8829476	62
29	0,4848095	0,8746195	61
30	0,4999997	0,8660254	60
31	0,5150378	0,8571675	59
32	0,5299192	0,8480477	58
33	0,5446389	0,8386706	57
34	0,5591929	0,8290377	56
35	0,5735764	0,8191517	55
36	0,5877648	0,8090174	54
37	0,6018146	0,7986356	53
38	0,615661	0,788011	52
39	0,62932	0,7771464	51
40	0,6427873	0,7660444	50
41	0,6560587	0,7547098	49
42	0,6691303	0,7431453	48
43	0,6819982	0,7313536	47
44	0,6946582	0,7193399	46
45	0,7071072	0,7071072	45

```
LIST
0005    LET P1=3.141593
0010    PRINT "*********************************";
0011    PRINT "*********************************"
0020    PRINT "*    ARC        I    SINUS      ";
0021    PRINT "I    COSINUS    I    ARC        *"
0030    PRINT "*********************************";
0031    PRINT "*********************************"
0040    FOR I=0 TO 45
0050    PRINT "*     ";I,"I";SIN(I*P1/180),
0051    PRINT "I";COS(I*P1/180),"I     ";90-I,"*"
0060    NEXT I
0070    PRINT "*********************************";
0071    PRINT "*********************************"
9999    END
```

VIII-3-5 - Génération d'une table de sinus/cosinus

```
*********************************
*        X      *    SQRT(X)    *
*********************************
*        1      * 1             *
*        2      * 1,414214      *
*        3      * 1,732051      *
*        4      * 2             *
*        5      * 2,236068      *
*        6      * 2,44949       *
*        7      * 2,645752      *
*        8      * 2,828427      *
*        9      * 3             *
*        10     * 3,162278      *
*        11     * 3,316625      *
*        12     * 3,464102      *
*        13     * 3,605552      *
*        14     * 3,741658      *
*        15     * 3,872984      *
*        16     * 4             *
*        17     * 4,123106      *
*        18     * 4,242641      *
*        19     * 4,358899      *
*        20     * 4,472136      *
*********************************
```

```
LIST
0010    PRINT "*********************************"
0020    PRINT "*         X      *    SQRT(X)    *"
0030    PRINT "*********************************"
0040    FOR I=1 TO 20
0050    PRINT "*      ";I,"*";SQR(I),"*"
0060    NEXT I
0070    PRINT "*********************************"
9999    END
```

VIII-3-6 - Génération d'une table de racines carrées

```
NUMBER          SQUARE ROOT     CUBE ROOT       FOURTH ROOT
 1               1               1               1
 2               1,414213        1,259921        1,189207
 3               1,732052        1,44225         1,316074
 4               2,000001        1,587402        1,414213
 5               2,236068        1,709976        1,495349
 6               2,449491        1,817121        1,565084
 7               2,645752        1,912933        1,626577
 8               2,828429        2,000001        1,681793
 9               3,000003        2,080085        1,732052
 10              3,162281        2,154436        1,77828
```

LIST

```
0010    PRINT "NUMBER","SQUARE ROOT","CUBE ROOT","FOURTH ROOT"
0020    FOR N=1 TO 10
0030       PRINT N,
0040       FOR R=2 TO 4
0050          PRINT N↑(1/R),
0060       NEXT R
0070       PRINT
0080    NEXT N
9999    END
```

VIII-3-7 - Calcul des racines carrée, cubique et quatrième

```
 1              2              3

 4              5              6

 7              8              9


A(3,3) = 35
```

LIST

```
0010    DIM A(3,3)
0020    MATREAD A
0030    MATPRINT A
0040    DEF FNA(X)=X*X+A(1,1)
0050    LET A(3,3)=A(3,1)*A(FNA(A(1,2)-3),FNA(SQR(A(1,1))))
0060    PRINT "A(3,3) =";A(3,3)
0100    STOP
1000    DATA 1,2,3,4,5,6,7,8,9
9999    END
```

VIII-3-8 - Modification d'un élément de matrice

```
4.000002        9.000017        16.00005        24.99999        36.00007

9.000017        16.00005        24.99999        36.00007        49.00011

0               0               0               0               0

0               0               0               0               0

0               0               0               0               0
```

LIST

```
0010    DIM A(5,5)
0020    GOSUB 45
0025    MATPRINT A,
0030    STOP
0045    FOR I=1TO 2
0050    FOR J=1 TO 5
0065    LET A(I,J)=(I+J)↑2
0070    NEXT J
0080    NEXT I
0090    RETURN
9999    END
```

VIII-3-9 - Initialisation d'un sous-tableau

```
1               2               3

4               5               6

7               8               9


1               2               3

4               5               6

7               8               9
```

LIST

```
0010    DIM A(3,3),B(3,3)
0020    MATREAD A(3,3)
0030    MATPRINT A
0040    MAT B=A
0050    MATPRINT B
0070    STOP
1000    DATA 1,2,3,4,5,6,7,8,9
9999    END
```

VIII-3-10 - Affectation matricielle

```
 1          2          3
 4          5          6
 7          8          9

 5          10         15
 20         25         30
 35         40         45

 6          12         18
 24         30         36
 42         48         54

-4         -8         -12
-16        -20        -24
-28        -32        -36
```

```
                                LIST
0010    DIM A(3,3),B(3,3),C(3,3)
0020    MATREAD A
0030    MATPRINT A
0040    PRINT
0050    MAT B=(5)*A
0060    MATPRINT B
0080    PRINT
0090    MAT C=A+B
0100    MATPRINT C
0110    PRINT
0120    MAT C=A-B
0130    MATPRINT C
1000    DATA 1,2,3,4,5,6,7,8,9
9999    END
```

VIII-3-11 - Addition - Soustraction matricielle
Multiplication par une constante

```
 2              4              6

 8             10             12


A*B=

-2             -2             -3

-2             -5             -9
```

```
                                LIST
0010    DIM A(10,10),B(10,10),C(10,10)
0020    READ M, N
0030    MATREAD A(M,N), B(N,N)
0040    MAT C=ZER(M,N)
0100    MAT C=A+A
0120    MATPRINT C
0140    MAT C=A*B
0150    PRINT "A*B="
0160    PRINT
0170    MATPRINT C
0190    DATA 2,3
0191    DATA 1,2,3
0192    DATA 4,5,6
0193    DATA 1,0,-1
0194    DATA 0,-1,-1
0195    DATA -1,0,0
0199    END
```

VIII-3-12 - Multiplication matricielle

```
SOMME:

 4              2              13

 5             -3               0


DIFFERENCE:

-2              6               1

 1              7               2


MULTIPLICATION PAR UN SCALAIRE:

 0.25           1               1.75

 0.75           0.5             0.25
```

```
                              LIST

0010    DIM A(2,3),B(2,3),D(2,3),K(2,3),S(2,3)
0100    MATREAD A(2,3),B(2,3)
0130    MAT S=A+B
0140    PRINT "SOMME:"
0145    PRINT
0150    MATPRINT S
0170    MAT D=A-B
0180    PRINT "DIFFERENCE:"
0185    PRINT
0190    MATPRINT D
0210    MAT K=(1/4)*A
0220    PRINT "MULTIPLICATION PAR UN SCALAIRE:"
0225    PRINT
0230    MATPRINT K
0900    DATA 1,4,7,3,2,1,3,-2,6,2,-5,-1
9999    END
```

```
5,6             6,5             8,02
2,1             3,567           5,403
2,9876          8,5             1

1               2               3               4               5
6               7               8               9               10
101             102             103             104             105
106             107             108             109             110

5               6               9               12              15
18              21              24              27              30
303             306             309             312             315
318             321             324             327             330
```

LIST

```
0010    DIM A(5,5)
0020    MATINPUT A(3,3)
0030    MATPRINT A
0040    MATINPUT A(A(1,1),5)
0050    MATPRINT A
0060    MAT A=(A(1,3))*A
0070    MATPRINT A
9999    END
```

VIII-3-14 - Entrée directe d'une matrice

0	0	0		
0	0	0		
0	0	0		
1	1	1		
1	1	1		
1	1	1		
1	0	0		
0	1	0		
0	0	1		
0	0	0	0	0
0	0	0		
0	0	0	0	0
0	0	0		
0	0	0	0	
0	0	0		
0	0	0	0	0
0	0	0		
0	0	0	0	0
0	0	0		
0	0	0	0	0
0	0	0		
0	0	0	0	0
0	0	0		
0	0	0	0	0
0	0	0		
0	0	0	0	0

1	1	1	1	1
1	1	1		
1	1	1	1	1
1	1	1		
1	1	1	1	1
1	1	1		
1	1	1	1	1
1	1	1		
1	1	1	1	1
1	1	1		
1	1	1	1	1
1	1	1		
1	1	1	1	1
1	1	1		
1	1	1	1	1
1	1	1		

1	0	0	0	0
0	0	0		
0	1	0	0	0
0	0	0		
0	0	1	0	0
0	0	0		
0	0	0	1	0
0	0	0		
0	0	0	0	1
0	0	0		
0	0	0	0	0
1	0	0		
0	0	0	0	0
0	1	0		
0	0	0	0	0
0	0	1		

```
LIST

0010    DIMA(10,10)
0020    MAT A=ZER(3,3)
0030    MATPRINT A
0040    MAT A=CON(3,3)
0050    MATPRINT A
0060    MAT A=IDN (3,3)
0070    MATPRINT A
0090    MAT A=ZER(8,8)
0100    MATPRINT A
0110    MAT A=CON
0120    MATPRINT A
0130    MAT A=IDN
0140    MATPRINT A
9999    END
```

VIII-3-15 - Initialisations matricielles

```
10          20          30          40          50
60          70          80

500         600         900         1200        1500
1800        2100        2400
```

```
LIST

0010    DIM A(1,8),B(1,8)
0020    MATINPUT A
0030    MAT B =(A(1,3))*A
0040    MATPRINT A,B
9999    END
```

VIII-3-16 - Opération matricielle

```
 1
 5.67E-06
 5
 5.E-03
 5
 6
 100000
 8
 21

 1     2     3     4     5     6     7     8     9
 10    11    12    13    14    15    16    17    18
 19    20    21    22    23    24    25    26    27
```

LIST

```
0010    DIM A(10),B(10,10)
0015    LET M=4
0020    LET N=7
0030    MATREAD A(N+(M/2))
0040    INPUT A(M),A(N),A(2)
0050    RESTORE 1020
0060    READ A((N-M)↑(2*A(1)))
0065    RESTORE 1000
0070    MATREAD B(3,9)
0080    MATPRINT A,B;
1000    DATA 1,2,3,4,5,6,7,8,9,10
1010    DATA 11,12,13,14,15,16,17,18,19,20
1020    DATA 21,22,23,24,25,26,27,28,29,30
9999    END
```

VIII-3-17 - Lecture de matrices dans les DATA

```
 1            2            3
 4            5            6
 7            8            9

 2            2            2
 0            0            0
 1            2            3

 5            8           11
14           20           26
23           32           41

24           30           36
 0            0            0
50           36           42
```

```
                          LIST
0010    DIM A(3,3),B(3,3),C(3,3)
0020    MATREAD A,B
0030    MATPRINT A,B
0035    PRINT
0040    MAT C=A*B
0050    MATPRINT C
0060    PRINT
0070    MAT C=B*A
0080    MATPRINT C
1000    DATA 1,2,3,4,5,6,7,8,9
1010    DATA 2,2,2,0,0,0,1,2,3
9999    END
```

VIII-3-18 - Multiplication matricielle dans les 2 sens

```
PUISSANCE : 1,999999

 0,4375          0,1875          0,375

 0,375           0,25            0,375

 0,375           0,1875          0,4375


PUISSANCE : 4,000002

 0,4023438       0,1992188       0,3984375

 0,3984375       0,203125        0,3984375

 0,3984375       0,1992188       0,4023438


PUISSANCE : 7,999997

 0,4000091       0,199997        0,399994

 0,3999939       0,2000122       0,3999939

 0,399994        0,1999971       0,4000091


PUISSANCE : 16

 0,4000001       0,2000001       0,4

 0,4000001       0,2000001       0,4000001

 0,4             0,2000002       0,4000001
```

LIST

```
0050    DIM A(5,5),M(5,5)
0100    READ N
0120    MATREAD M(N,N)
0125    MAT A=ZER(N,N)
0130    FOR I=1 TO 4
0140    PRINT "PUISSANCE :";2↑I
0145    PRINT
0150    MAT A=M*M
0160    MATPRINT A
0170    MAT M=A
0180    NEXT I
0190    DATA 3
0191    DATA ,5,,25,,25,,5,0,,5,,25,,25,,5
9999    END
```

VIII-3-19 - Elévation d'une matrice à une puissance

```
1
0
2

1          2          3
4          5          6
7          8          9

1
16
25
```

```
LIST
0010   DIM A(3,1),B(3,3),C(3,1)
0020   MATREAD A
0030   MATREAD B
0040   MAT C=B*A
0050   MATPRINT A,B,C
1000   DATA 1,0,2
1010   DATA 1,2,3,4,5,6,7,8,9
9999   END
```

```
1          2          3
4
5
6
```

VIII-3-20 - Multiplication d'une matrice par un vecteur-colonne

```
1
2
3
```

VIII-3-21 - Transposition de matrice

```
LIST
0010   DIM A(1,3),B(3,1)
0020   MATREAD A
0030   MATREAD B
0040   MATPRINT A,B
0050   MAT B=TRN(A)
0060   MATPRINT B
1000   DATA 1,2,3,4,5,6
9999   END
```

TOUT A ZERO :

0 0 0

0 0 0

0 0 0

TOUT A UN :

1 1 1

1 1 1

1 1 1

MATRICE IDENTIQUE :

1 0 0

0 1 0

0 0 1

MATRICE CARREE :

1	0,5	0,3333333
0,5	0,3333333	0,25
0,3333333	0,25	0,2

SON INVERSE :

9,000173	-36,00095	30,00092
-36,00096	192,0052	-180,005
30,00092	-180,005	180,0048

ERREURS D'ARRONDI :

3,E-06	-1,E-05	1,E-05
-3,E-06	1,E-05	0
1,E-06	-2,E-05	2,E-05

LIST

```
0050    DIM A(5,5),B(5,5),C(5,5),R(5,5)
0100    MAT A=ZER(3,3)
0110    MAT B=CON(3,3)
0120    MAT C=IDN(3,3)
0130    MAT R=ZER(3,3)
0200    PRINT "TOUT A ZERO :"
0205    PRINT
0210    MATPRINT A;
0220    PRINT "TOUT A UN :"
0225    PRINT
0230    MATPRINT B;
0240    PRINT "MATRICE IDENTIQUE :"
0245    PRINT
0250    MATPRINT C;
0300    MATREAD A
0310    PRINT "MATRICE CARREE :"
0315    PRINT
0320    MATPRINT A
0330    MAT B=INV(A)
0340    PRINT "SON INVERSE :"
0345    PRINT
0350    MATPRINT B
0400    MAT R=A*B
0410    MAT R=R-C
0420    PRINT "ERREURS D'ARRONDI :"
0425    PRINT
0430    MATPRINT R
0900    DATA 1,.5,.3333333
0910    DATA .5,.3333333,.25
0920    DATA .3333333,.25,.2
9999    END
```

VIII-3-22 - Inversion d'une matrice de Hilbert de rang 3

MATRICE INITIALE

1,2	3,4	5,6
5,6	1,2	3,4
3,4	5,6	1,2

MATRICE INVERSEE

-0,1186554	0,1841949	3,267973E-02
3,267975E-02	-0,1188354	0,1841949
0,1841949	3,267972E-02	-0,1188354

ERREURS D'ARRONDI : A*INV(A)-IDN

-3,E-07	-1,E-07	2,E-07
4,E-07	-5,E-07	0
1,E-07	1,6E-07	-5,E-07

ERREURS D'ARRONDI : INV(A)*A-IDN

-4,E-07	0	1,8E-07
4,E-07	-3,E-07	1,E-07
-1,E-07	2,E-07	-5,E-07

INVERSE DE L'INVERSE

1,2	3,4	5,6
5,599999	1,2	3,4
3,399999	5,600001	1,2

ERREURS D'ARRONDI : INV(INV(A))-A

0	0	0
1,E-06	0	0
1,E-06	-1,E-06	0

MATRICE INITIALE

1	0	0
0	1	0
0	0	1

MATRICE INVERSEE

1	0	0
0	1	0
0	0	1

ERREURS D'ARRONDI : A*INV(A)-IDN

0	0	0
0	0	0
0	0	0

ERREURS D'ARRONDI : INV(A)*A-IDN

0	0	0
0	0	0
0	0	0

INVERSE DE L'INVERSE

1	0	0
0	1	0
0	0	1

ERREURS D'ARRONDI : INV(INV(A))-A

0	0	0
0	0	0
0	0	0

LIST

```
0010    DIM A(10,10),B(10,10),C(10,10)
0015    DIM D(10,10)
0020    INPUT N
0030    MATINPUT A(N,N)
0040    MAT B=ZER(N,N)
0050    MAT C=ZER(N,N)
0055    MAT D=IDN(N,N)
0060    PRINT "MATRICE INITIALE"
0070    PRINT
0075    MATPRINT A
0080    MAT B=INV(A)
0090    PRINT "MATRICE INVERSEE"
0100    PRINT
0110    MATPRINT B
0120    MAT C=A*B
0130    MAT C=C-D
0140    PRINT "ERREURS D'ARRONDI : A*INV(A)-IDN"
0150    PRINT
0160    MATPRINT C
0170    MAT C=B*A
0180    MAT C=C-D
0190    PRINT "ERREURS D'ARRONDI : INV(A)*A-IDN"
0200    PRINT
0210    MATPRINT C
0220    MAT C=INV(B)
0230    PRINT "INVERSE DE L'INVERSE"
0240    PRINT
0250    MATPRINT C
0260    MAT B=A-C
0270    PRINT "ERREURS D'ARRONDI : INV(INV(A))-A"
0280    PRINT
0290    MATPRINT B
0300    INPUT X
0310    IF X<> 0 THEN 20
9999    END
```

VIII-3-23 - Inversions successives - Recherche des erreurs d'arrondi

CONCLUSION

Le travail réalisé tout au long de cette étude et décrit dans le présent exposé, montre bien qu'il est possible d'équiper une petite machine comme le GE 55 d'un compilateur pour un langage assez puissant tel le BASIC.

On peut même dire que la conception du GE 55 orientée vers la gestion de stocks et la facturation, facilite l'emploi d'un langage conversationnel, dans la mesure où justement la machine est prévue pour l'intervention de l'opérateur au cours de l'exécution d'un programme.

Malheureusement, les performances médiocres de cette machine font regretter que l'application de cette étude n'ait pas été faite sur un ordinateur plus performant. A cet égard, la transposition du compilateur du GE 55 au GE 58, de même taille, environ 8 fois plus rapide, équipé d'un système-disque, peut se révéler intéressante.

Dans le cadre strict du GE 55, on peut penser à améliorer les performances du compilateur BASIC. En effet, l'étude terminée, on connaît exactement l'encombrement de toutes les tables et des différentes zones. On sait d'autre part que les causes principales de ralentissement, sont d'une part les changements de segments, d'autre part les échanges entre zones de mémoire et pistes du tambour. Il est bien clair qu'une étude statistique sur les segments permet de savoir quels sont ceux qui viennent le plus fréquemment en mémoire centrale. On peut alors penser qu'une segmentation réalisée avec des segments moins longs et regroupant dans des segments uniques, les modules utilisés fréquemment, permettrait d'une part d'avoir en permanence en mémoire les tables ou piles que faute de place, on renvoie en tambour, limitant ainsi les échanges avec le

tambour aux seuls blocs de programmes et matrices, et d'autre part de gagner globalement un temps non négligeable sur les changements de segments.

Une étude rapide du traitement des mots-clés, envisagé avec la liste enchaînée résidant en mémoire, et des segments d'une piste, permet d'obtenir une durée moyenne de 570 ms contre 1115 avant. Soit un gain égal à 2 approximativement. Il est difficile d'obtenir le facteur de gain global sur le traitement de tout un programme, mais il est bien clair qu'il n'est pas négligeable.

Malheureusement, ne disposant plus de la machine, il a été impossible de faire ces améliorations au niveau de la programmation.

Une autre étape vers un produit commercialisable, consiste à fondre en un seul, les programmes d'analyse syntaxique et d'interprétation, de façon à rendre plus souple la mise en oeuvre sur machine. Il suffit en phase d'analyse, de sauter l'exécution des modules d'interprétation. Cette méthode est facilitée par le fait que l'interprétation s'appuie sur l'algorithme d'analyse.

Nous pouvons remarquer que l'étude faite est applicable quelle que soit la taille de la machine, les modifications à apporter relatives à une mémoire centrale plus grande, ne sont en fait que des simplifications, et qui permettent d'éviter tous les échanges avec un support magnétique quelconque, apportant ainsi des gains de temps considérables.

Enfin, la seule modification des modules sémantiques permet de passer d'un compilateur interprétatif à un compilateur génératif, la méthode d'analyse employée reste toujours valable dans ce cas.

La systématisation des méthodes employées ici, à des compilateurs divers, nous conduit à penser à la génération automatique de l'analyseur, à partir des règles de grammaire.

La conception d'un compilateur de compilateur est une chose ardue, mais on peut penser que l'emploi des matrices booléennes représentant la grammaire apporterait ici un point de départ valable pour une telle conception.

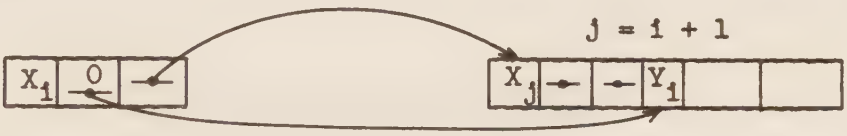
ANNEXE A

LISTE ENCHAINEE DES MOTS-CLES

A-1 - PRINCIPES

La consultation de la liste enchaînée des mots-clés est un moyen commode pour la reconnaissance des mots-clés.

La reconnaissance se fait lettre par lettre, en commençant par la première. Le test sur une lettre fait appel à la consultation d'une cellule du type :



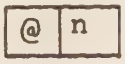
Mot-clé $X_1X_2 \ldots X_iX_j \ldots X_n$

Figure A-1

Le premier octet contient la lettre considérée, le troisième champ contient un pointeur sur la lettre suivante du mot-clé courant, on ne l'utilise que si l'on a identifié la même lettre que celle qui figure en tête ; si ce n'est pas le cas, on exploite le pointeur contenu dans le deuxième champ, qui permet d'avoir l'adresse d'une cellule du même type comportant une autre lettre (de même rang dans le mot-clé). Si le deuxième champ contient 0, c'est qu'il n'y a pas de mot-clé correspondant à la chaîne de lettres qu'on présente en entrée, il y a alors une sortie en erreur.

Cette méthode est intéressante à deux titres, d'abord quand plusieurs mots-clés ont une partie initiale commune ($\forall$ nombre de lettres communes), ce sont les mêmes cellules (de tête) qui

servent, et d'autre part cette méthode est utilisable sans problème pour des mots-clés de longueur variable, il suffit de créer dans la liste une cellule d'achèvement de mot-clé.



@ : caractère d'achèvement
n : code du mot-clé

Figure A-2

La recherche systématique en tête de cellule du caractère d'achèvement permet de déterminer l'arrêt de l'analyse du mot-clé. Le code obtenu dans la cellule d'achèvement permet d'exploiter une table de branchement aux routines de traitement des mots-clés.

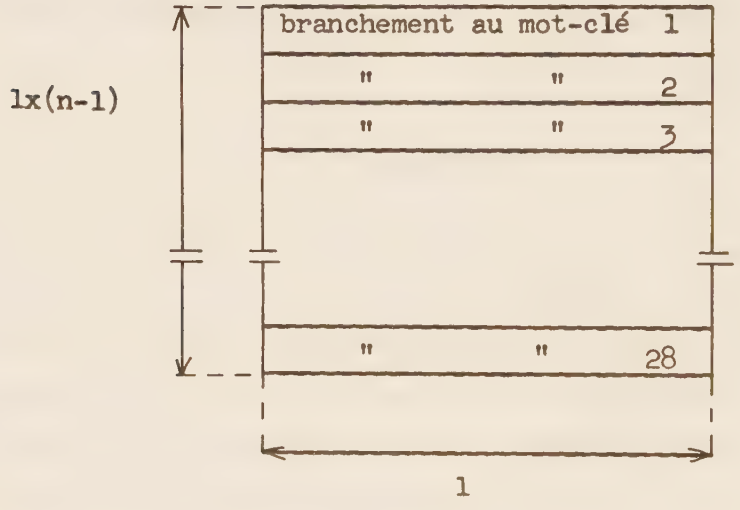


Figure A-3

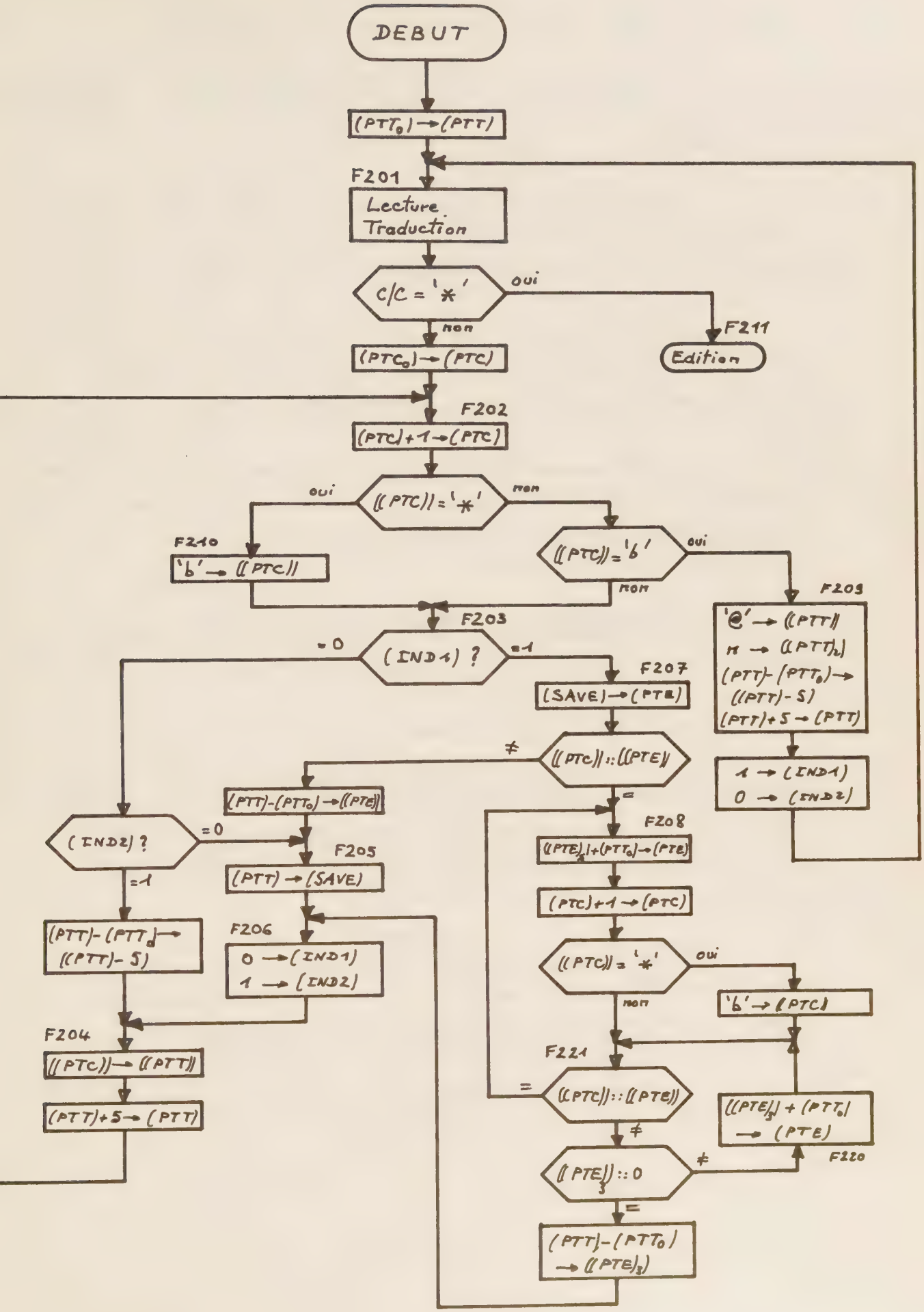

DEBUT
(PTT0) → (PTT)
F201
Lecture Traduction
C/C = '*'
oui
non
F211
Edition
(PTC0) → (PTC)
F202
(PTC)+1 → (PTC)
((PTC)) = '*'
oui
non
F210
'b' → ((PTC))
((PTC)) = 'b'
oui
non
F209
'@' → ((PTT))
M → ((PTT)2)
(PTT)-(PTT0) → ((PTT)-5)
(PTT)+5 → (PTT)
1 → (IND1)
0 → (IND2)
F203
(IND1) ?
=0
=1
F207
(SAVE) → (PTE)
((PTC)) :: ((PTE))
≠
=
(PTT)-(PTT0) → ((PTE))
F208
((PTE)2)+(PTT0) → (PTE)
(PTC)+1 → (PTC)
((PTC)) = '*'
oui
non
'b' → (PTC)
F221
((PTC)) :: ((PTE))
=
≠
((PTE)3) :: 0
≠
=
((PTE)3) + (PTT0) → (PTE)
F220
(PTT)-(PTT0) → ((PTE)3)
(IND2) ?
=0
=1
F205
(PTT) → (SAVE)
F206
0 → (IND1)
1 → (IND2)
(PTT)-(PTTd) → ((PTT)-5)
F204
((PTC)) → ((PTT))
(PTT)+5 → (PTT)

Figure A-4

A-2 - CREATION

Les cartes comportant les mots-clés et leurs codes associés sont présentées dans le lecteur de cartes de façon à ce que les mots-clés soient dans l'ordre alphabétique.

PTC est le pointeur de caractère dans la zone de lecture. PTT est le pointeur de travail dans la liste. PTE est le pointeur d'enchaînement des mots-clés.

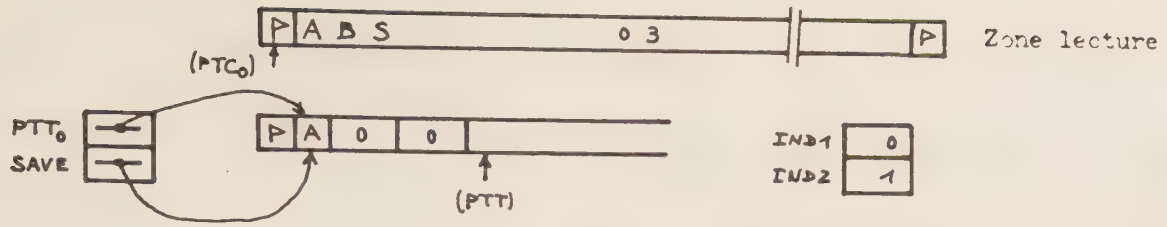
L'enchaînement est réalisé quand IND1 est à 1, c'est-à-dire au début du traitement de chaque mot-clé (sauf pour le premier). L'enregistrement normal des lettres d'un mot-clé est réalisé quand IND1 est à 0 et IND2 à 1.

La détection du premier blanc indique que le mot-clé est terminé et il faut alors créer la cellule d'achèvement.

Dans certains cas, il faut que le mot-clé comporte un blanc dans la liste, c'est particulièrement le cas pour MAT, car il faut distinguer ce mot-clé des autres mots-clés MATINPUT, MATPRINT, et MATREAD. Pour cela, on introduit dans la chaîne d'entrée, un astérisque à l'endroit où l'on veut un blanc, cet astérisque est remplacé par un blanc dans la liste.

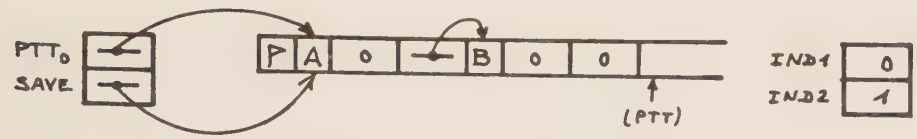
La détection de ' * ' en code-carte permet de se débrancher vers la routine d'édition qui permet l'obtention de la liste sous forme de cartes constantes tambour exploitable par l'assembleur.

Les figures A-5 montrent l'état de la liste pendant sa création.



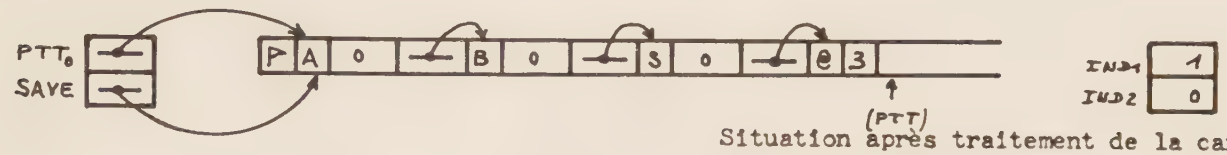
Situation après prise en charge du 1er caractère de la 1ère carte

Figure A-5 (a)



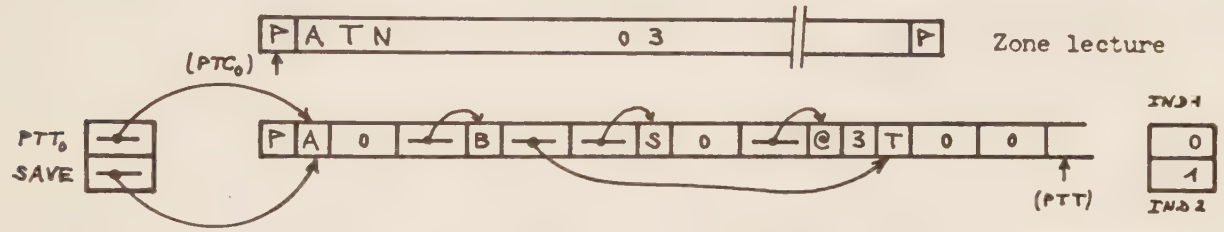
Situation au retour en F502 avant prise en charge de S

Figure A-5 (b)



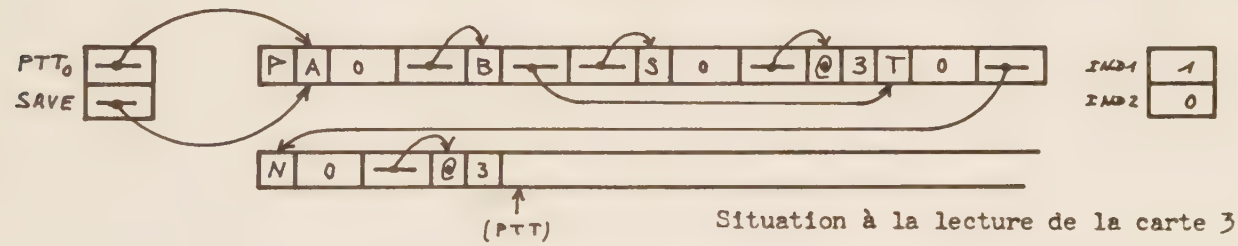
Situation après traitement de la carte au retour en F501 avant lecture carte 2

Figure A-5 (c)



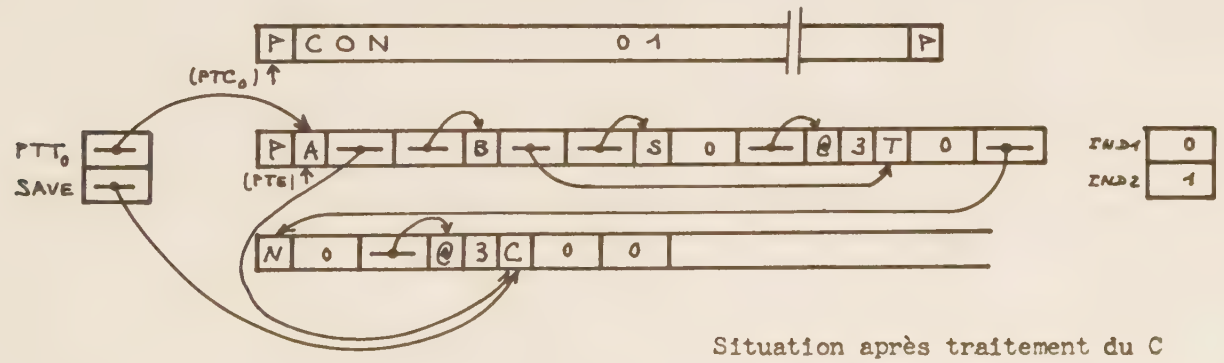
Situation après traitement de AT, T pris en charge dans F508

Figure A-5 (d)



Situation à la lecture de la carte 3

Figure A-5 (e)



Situation après traitement du C

Figure A-5 (f)

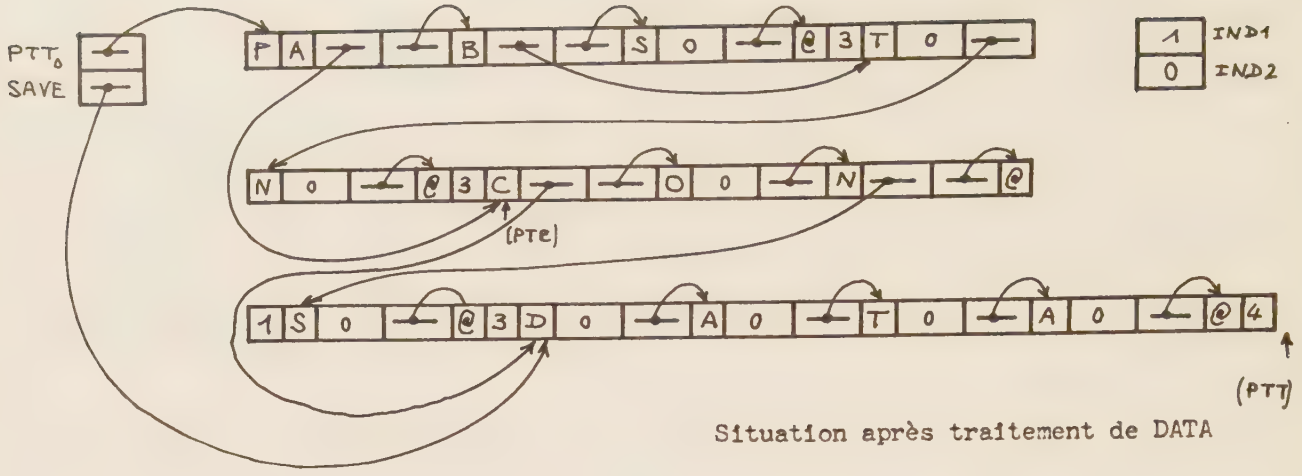


Figure A-5 (g)

A-3 - UTILISATION

Dans l'analyse et dans l'interprétation, on utilise la liste, en vue d'obtenir le code associé au mot-clé courant, ou de déterminer que la suite de caractères qu'on analyse n'est pas un des mots-clés autorisés.

La consultation de la liste enchaînée (figure 2-6) commence par l'appel de liste en mémoire, suivi de l'initialisation d'un pointeur de travail PTT sur la tête de la première cellule. On tourne dans la boucle A215, jusqu'à la reconnaissance du premier caractère du mot-clé, on arrive alors en A220 où l'on fait progresser PTT de façon à pointer sur le caractère suivant du mot-clé ; il y a prise en charge du caractère suivant de la chaîne d'entrée et l'on retourne dans la boucle de recherche pour le second caractère, le processus se répète pour toutes les lettres jusqu'à la détection du caractère d'achèvement. On prélève alors le code associé au mot-clé et l'on s'en sert pour se brancher à la routine de traitement du mot-clé considéré.

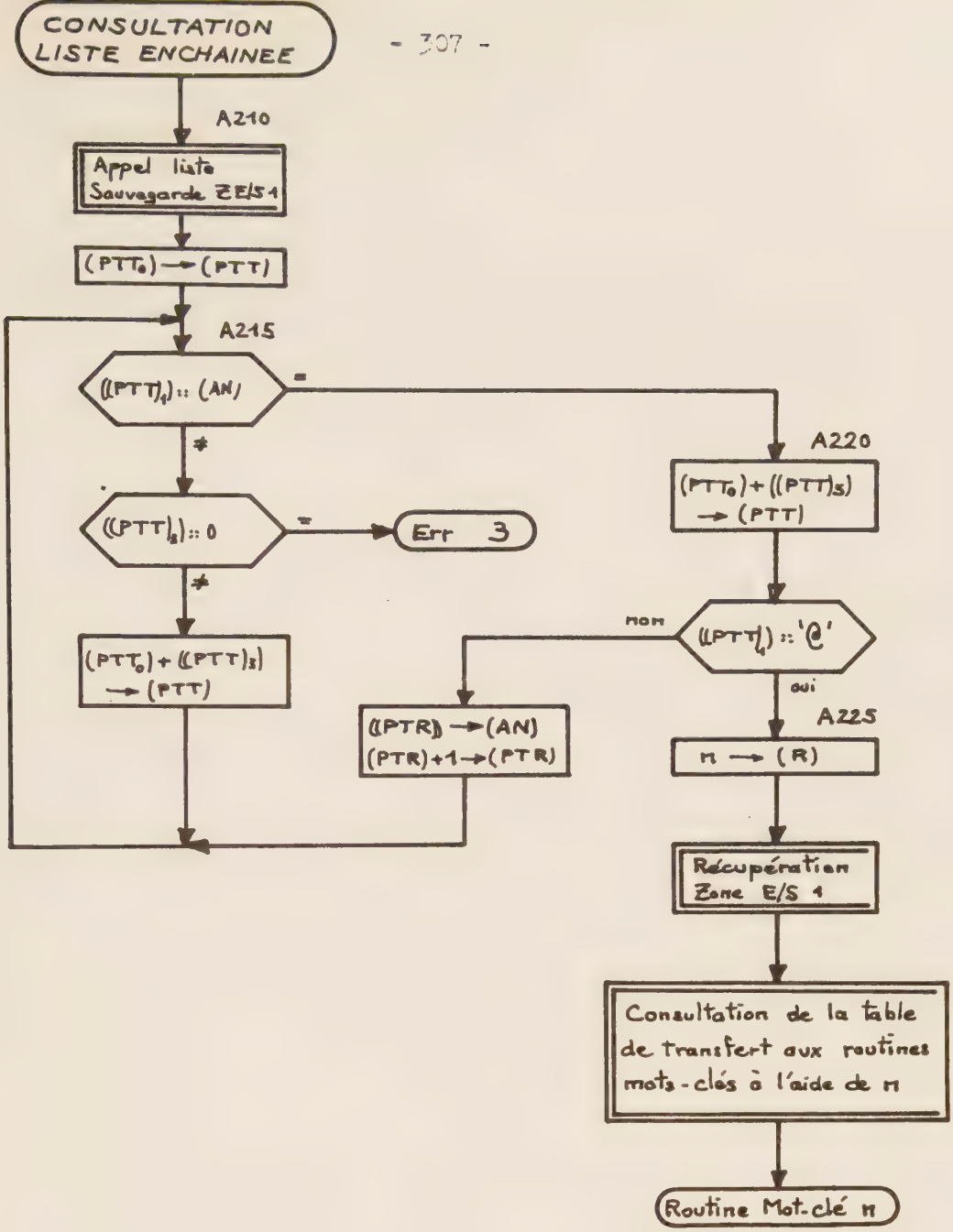

CONSULTATION LISTE ENCHAINEE
A210
Appel liste Sauvegarde ZE/S 1
(PTT0) → (PTT)
A215
((PTT)1) :: (AN)
=
≠
A220
(PTT0) + ((PTT)3) → (PTT)
((PTT)2) :: 0
=
Err 3
≠
non
((PTT)1) :: '@'
oui
(PTT0) + ((PTT)3) → (PTT)
((PTR)) → (AN)
(PTR) + 1 → (PTR)
A225
n → (R)
Récupération Zone E/S 1
Consultation de la table de transfert aux routines mots-clés à l'aide de n
Routine Mot-clé n

Figure A-6

La détection de 0 dans le second champ d'une cellule indique qu'il n'y a pas reconnaissance de la lettre courante et qu'il faut sortir en erreur.

```
L.CN.NUMB.  ADD.OC.     CODING

15          0301 BEGIN  0301
12 03       0301 MRR    E12101          PTTO DANS PTT
14 02       0304 LEVEL  F201                 LECTURE MOT-CLE
12 04       0304 IOC    53019606        LECTURE
12 02       0308 TR1    5106            TRADUCTION
12 06       0310 JIERT  042A6001F111    SI FIN ALLER EN EDITION
12 03       0316 MRR    E10603          BASE LECTURE DANS PTC
14 02       0319 LEVEL  F202                 CARACTERE SUIVANT
12 03       0319 ADD    A10503          PTC PLUS 1 DANS PTC
12 06       0322 JIERT  042A3000F110
12 06       0328 JIERT  04203000F109    SI BLANC ALLER EN 9
14 02       0334 LEVEL  F203
12 06       0334 JIERT  04010251F107    TEST IND1
12 06       0340 JIURT  05010252F105    TEST IND2
12 03       0346 MRR    E10125
12 03       0349 SBD    B12125
12 06       0352 MVC    800202251000    PTT-PTTO DANS CELLULE PRECEDENTE
14 02       0358 LEVEL  F204
12 06       0358 MVC    800130001001    CARACTERE DANS CELLULE COURANTE
12 03       0364 ADD    A12301          PTT PLUS 5 DANS PTT
12 03       0367 JRT    02F102          RETOUR CARACTERE SUIVANT
14 02       0370 LEVEL  F205
12 03       0370 MRR    E10122          PTT DANS SAVE
14 02       0373 LEVEL  F206
12 04       0373 NI     61000251        0 DANS IND1
12 04       0377 OI     63010252        1 DANS IND2
12 03       0381 JRT    02F104
14 02       0384 LEVEL  F207
12 03       0384 MRR    E12204          SAVE DANS PTE
12 06       0387 CMC    880130004001    CARACTERE COMPARE AVEC CARACTERE
12 06       0393 JIERT  04300086F108    POINTE PAR PTE
12 03       0399 MRR    E10125
12 03       0402 SBD    B12125
12 06       0405 MVC    800202254003    PTT-PTTO DANS ZONE ENCHAINEMENT
12 03       0411 JRT    02F105
14 02       0414 LEVEL  F208                 DEROULEMENT CHAINAGE
12 03       0414 LRN    6002400504
12 03       0419 ADD    A12104
```

```
C.CN.NUMB. ADD.OC.     CODING

12 05      0422 ADD    A10503
12 06      0425 JIURT  052A3000F121
12 04      0431 NI     61203000
14 02      0435 LEVEL  F221
12 06      0435 CMC    880130004001
12 06      0441 JIERT  043D0086F108
12 06      0447 CMC    880240030100
12 06      0453 JIURT  053D0086F120
12 05      0459 MRR    E10125
12 05      0462 SBD    B12125
12 06      0465 MVC    800202254003
12 05      0471 JRT    02F106
14 02      0474 LEVEL  F220
12 05      0474 LRN    6002400304
12 05      0479 ADD    A12104
12 05      0482 JRT    02F121
14 02      0485 LEVEL  F209
12 04      0485 OI     63401001
12 06      0489 PKH    840260111002
12 05      0495 MRR    E10125
12 05      0498 SBD    B12125
12 06      0501 MVC    800202251000
12 05      0507 ADD    A12401
12 04      0513 OI     63010251
12 04      0514 NI     61000252
12 05      0519 JRT    02F101
14 02      0521 LEVEL  F210
12 04      0521 NI     61203000
12 05      0525 JRT    02F103
14 02      0528 LEVEL  F211
12 06      0528 INC    822090809000
12 06      0534 MVC    800702799008
12 05      0540 MRR    E10526
12 05      0543 MRR    E12127
14 02      0546 LEVEL  F230
12 05      0546 MRR    E12702
12 05      0549 ADD    A12802
12 05      0552 MRR    E10103
```

CARACTERE COMPARE AVEC CARACTERE
POINTE PAR PTE
TEST ZONE ENCHAINEMENT
SI ZONE NON NULLE ALLER EN 20

PTT-PTTO DANS ZONE ENCHAINEMENT

DEROULEMENT CHAINAGE

@ DANS CELLULE FIN DE MOT-CLE
CONDENSATION NUMERO DE MOT-CLE

PTT-PTTO DANS CELLULE PRECEDENTE
PTT PLUS 2 DANS PTT
1 DANS IND1
0 DANS IND2
RETOUR LECTURE CARTE MOT-CLE

REMPLACEMENT * PAR BLANC

EDITION
MISE A BLANC ZONE PERFORATION
MISE EN PLACE C/C ET NO DE PISTE
1 DANS DEPLACEMENT
PTTO DANS POINTEUR
TRAITEMENT ITERATIF

RECHERCHE DIFFERENCE ENTRE HAUT DE

```
SEQ.NUMB. ADD.OC.   CODING

12 05     0555 SBD      B10203          TABLE ET POINTEUR AUGMENTE DE 30
12 06     0558 UPH      856020009076    ETALEMENT 30 OCTETS TABLE EN Z. PERF
12 06     0564 JIERT    04010253F140    SI IND3 EGALE 1 ALLER EN 40
12 05     0570 CMD      910328          DIFFERENCE COMPAREE A 30
12 06     0573 JIURT    05300086F135
12 04     0579 OI       63010253        1 DANS IND3
14 02     0583 LEVEL    F235
12 06     0583 UPH      850202409005    ETALER 30 EN ZONE PERFO
12 05     0589 ADD      A12827          POINTEUR PLUS 30 DANS POINTEUR
12 06     0592 UPH      850302309011    ETALEMENT COMPTEUR EN ZONE PERFO
12 05     0598 ADD      A12826
12 02     0601 TRB      5B09            TRADUCTION
12 04     0603 IOC      5301AA09        PERFORATION
12 05     0607 JRT      02F130          RETOUR TRAITEMENT ITERATIF
14 02     0610 LEVEL    F240                 DERNIERE EDITION
12 06     0610 UPH      850302309011    ETALEMENT COMPTEUR EN ZONE PERFO
12 05     0616 ADD      A12803
12 06     0619 UPH      850201159005    ETALEMENT DIFFERENCE EN ZONE PERFO
12 05     0625 MRR      E10902
12 05     0628 ADD      A10302
12 05     0631 ADD      A10302
12 05     0634 SBD      B10502
12 05     0637 ADD      A12902          CALCUL ADRESSE DERNIER CARACTERE
12 06     0640 INC      822090762000    MISE A BLANC ZONE INUTILE
12 02     0646 TRB      5B09            TRADUCTION
12 04     0648 IOC      5301AA09        PERFORATION
12 05     0652 ADD      A10326          COMPTEUR PLUS DIFFERENCE DANS COMPT.
12 06     0655 UPH      850302309011    ETALEMENT COMPTEUR EN ZONE PERFO
12 06     0661 UPH      850201259005    ETALEMENT 01 EN ZONE PERFO
12 06     0667 INC      822090769017
12 06     0673 MVC      800202729018    F4 EN ZONE PERFO
12 02     0679 TRB      5B09            TRADUCTION
12 04     0681 IOC      5301AA09        PERFORATION
12 01     0685 STOP     40              FIN
11 01     0686 END      F3
32 02     0204          1000            PTTO
32 01     0215          05              LONGUEUR CELLULE NORMALE
32 01     0220          02              LONGUEUR CELLULE FIN
```

```
C,CN,NUMB, ADD,OC,      CODING

32 01       0240        30
32 01       0245        17
31 07       0273        52  050          C/C ET NUMERO DE PISTE
31 02       0271        14
16          0686 0301
```

```
ABS         03
ATN         03
CON         01
COS         03
DATA        04
DEF         05
DIM         06
END         07
EXP         03
FOR         08
GOSUB       09
GOTO        10
IDN         01
IF          11
INPUT       12
INV         02
LET         13
LOG         03
MATINPUT    25
MATPRINT    26
MATREAD     27
MAT*        24
MAX         28
MIN         28
NEXT        14
PRINT       15
READ        16
REM         17
RESTORE     18
RETURN      19
SIN         03
SQR         03
STEP        20
STOP        21
TAB         03
THEN        22
TO          23
TRN         02
ZER         01
```

```
>2300050001    41002900C5420C1700105300000015400354000000224E000000274003 43
>2300050031    00530034 4F000000394E004600444001530000005140037440099005841 00
>2300050061    75006354000000684100000073400445008700804600000008540054900 00
>2300050091    00924D0000009740064501280104 4E011601094400000114400758000001
>2300050121    21500000012640034601450133 4F000001385200000143400847018401 50
>2300050151    4F000001555301720160550000016542000000170400954000001774F0000
>2300050181    01824010490237018944020101944E000001994001460207020640114E00
>2300050211    00021350023002185500000223540000003284012560000023540024C0266
>2300050241    02424502540247540000C25240134F000002594700000264400340038302
>2300050271    71410371027654036402814903080286 4E000002915000000296550000 03
>2300050301    01540000030640255003350313520000031849000003234E000003285400
>2300050331    00033340265203570340450000034541000000350440000035540272000 00
>2300050361    03624024580000036940284900000037 64E000003814028 4E040503884500
>2300050391    00039358000003985400000040340145004320410520000041549000004 20
>2300050421    4E000004255400000430401552051004374500000442410454044744000 0
>2300050451    04524016400461045940175304880466540000047 14F000004765200000 4
>2300050481    81450000048640185400000493550000049852000005034E000005084019
>2300050511    53056805154905270520 4E000005254003510539053252000005374003 54
>2300050541    00000544450556054950000005544020 4F000005615000000566402154 06
>2300050571    21057341058505784800000583400348040059045000005954E00000600
>2300050601    40224F060906074023520000006144E000006194002 5A000006264500000 6
>2300050631    315200000636400 1
>2300050639    F4
```

5.2.4 - Gravure du tambour et code H 14.012

N° ordre de frappe	Caractère	Code perfo H			Code interne ISO
1	b	-	-	-	20
2	0	-	0	-	30
3	1	-	1	-	31
4	2	-	2	-	32
5	3	-	3	-	33
6	4	-	4	-	34
7	5	-	5	-	35
8	6	-	6	-	36
9	7	-	7	-	37
10	8	-	8	-	38
11	9	-	9	-	39
12	.	12	8	3	2E
13	/	0	1	-	2F
14		-	8	3	23
15	%	0	8	4	25
16	-	11	-	-	2D
17	*	11	8	4	2A
18	A	12	1	-	41
19	B	12	2	-	42
20	C	12	3	-	43
21	D	12	4	-	44
22	E	12	5	-	45
23	F	12	6	-	46
24	G	12	7	-	47
25	H	12	8	-	48
26	I	12	9	-	49
27	J	11	1	-	4A
28	K	11	2	-	4B
29	L	11	3	-	4C
30	M	11	4	-	4D
31	N	11	5	-	4E
32	O	11	6	-	4F

N° ordre de frappe	Caractère	Code perfo H			Code interne ISO
33	P	11	7	-	50
34	Q	11	8	-	51
35	R	11	9	-	52
36	S	0	2	-	53
37	T	0	3	-	54
38	U	0	4	-	55
39	V	0	5	-	56
40	W	0	6	-	57
41	X	0	7	-	58
42	Y	0	8	-	59
43	Z	0	9	-	5A
44	$	11	8	3	24
45	]	12	8	4	5D
46	@	-	8	4	40
47	&	12	-	-	26
48	,	0	8	3	2C
49	- Vis -				
50	'	11	8	7	27
51	(	12	8	5	28
52	<	12	8	6	3C
53	\	12	8	7	5C
54	↑	11	8	-	5E
55	;	11	8	6	3B
56	=	0	8	5	3D
57	"	0	8	6	22
58	!	0	8	7	21
59	>	-	8	6	3E
60	?	-	8	7	3F
61	←	0	8	2	5F
62	:	-	8	5	3A
63	+	12	0	-	2B
64	[	-	8	2	5B
65	)	11	8	5	29

Caractères rares (symboles du code H14.012 moins symboles du code H7)

ANNEXE B

PROGRAMME DE TRI EN TAMBOUR

B-1 - NECESSITE D'UN TRI EN TAMBOUR

Nous avons vu qu'il est hors de question de conserver en mémoire centrale toutes les instructions d'un programme BASIC, il faut donc évacuer sur le tambour les instructions regroupées en blocs ayant approximativement une taille qui correspond à la place offerte par une piste.

Pour des raisons de facilité, on normalise la longueur du numéro d'une instruction à 4 chiffres, sachant qu'un numéro de ligne est obligatoire, il y a au moins un chiffre frappé par le programmeur sur une ligne de 72 caractères au maximum, avec la normalisation on peut obtenir des lignes de 75 caractères au plus.

Une piste du tambour comporte 700 caractères, la division de 700 par 75 nous amène à choisir des blocs de 9 instructions, soit 675 caractères au total par bloc.

Le système conversationnel offert par BASIC donne la possibilité de frapper les instructions du programme dans un ordre quelconque, le système rétablissant l'ordre numérique. D'autre part, on peut éliminer l'une quelconque des instructions déjà entrées par la frappe de son seul numéro de ligne, on peut également remplacer une ligne par une autre ayant le même numéro. Toutes ces contraintes et possibilités imposent d'avoir une séquence capable de rechercher en tambour, une instruction donnée, grâce à son numéro de ligne, puis d'effectuer les modifications demandées, quitte à redistribuer complètement les instructions sur les pistes.

Cette séquence constitue un programme de tri d'un type particulier puisqu'on admet qu'elle ne travaille qu'avec une seule donnée qui doit être insérée dans un ordre déjà existant.

B-2 - ENVIRONNEMENT DE LA SEQUENCE DE TRI

La séquence de tri s'intègre dans un algorithme plus vaste qui définit l'analyseur (voir § VII-2), il nous a semblé intéressant de donner les organigrammes des algorithmes qui traitent les autres séquences.

La figure B-2 montre l'organigramme de l'algorithme de simulation d'un télétype par le clavier alphanumérique du GE 55.

Il faut distinguer dans la ligne frappée, les ordres de commande (LIST et PUNCH), des lignes banales (voir figure B-3), il faut vérifier la validité du numéro de ligne (c'est ce qui explique que l'algorithme d'analyse ne s'attache pas au numéro de ligne qui précède le mot-clé), en F20A, on positionne à 1 l'indicateur AAA dans le cas où la ligne frappée ne comporte que des blancs (outre le numéro de ligne), ce qui signifie qu'il convient d'éliminer du programme déjà enregistré, la ligne de même numéro, AAA est à 0 dans le cas contraire, on ne se branche alors sur la séquence de tri que si l'instruction ne présente pas d'erreur de syntaxe.

B-3 - PRINCIPE DE L'ENREGISTREMENT DES BLOCS EN TAMBOUR

Pour faciliter la recherche des lignes dont le numéro se trouve en arrière, on conserve en permanence le bloc courant et son prédécesseur dans deux zones MC1 et MC2 contiguës en mémoire. Le bloc courant figure dans MC2. Les limites de ces blocs sont INIT1, FIN1, INIT2 et FIN2.

PMC est le pointeur de la cellule courante qui reçoit après normalisation (voir figure B-4 - F20C) la ligne qui figure dans ZL, la zone de lecture.

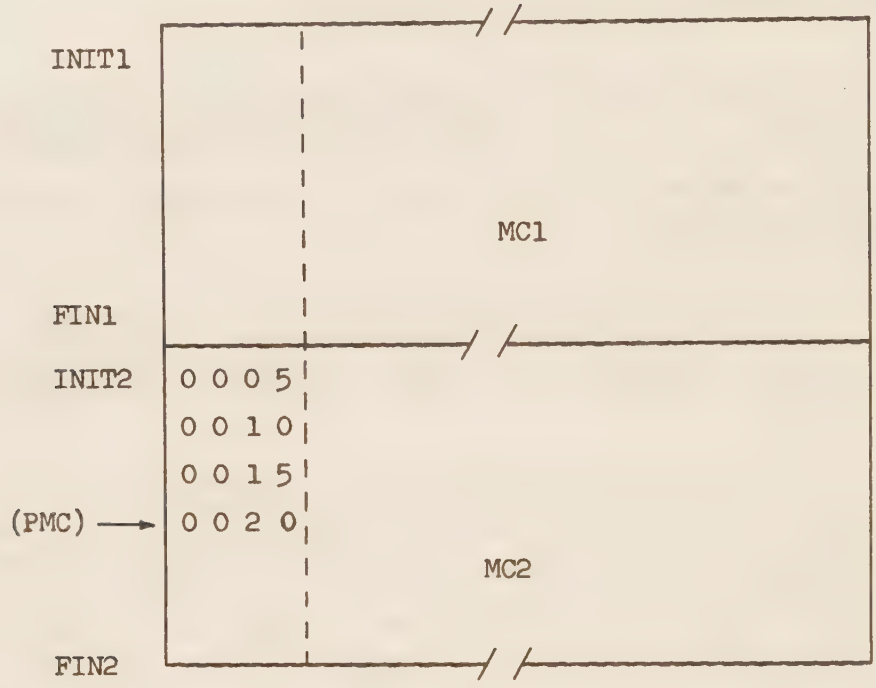
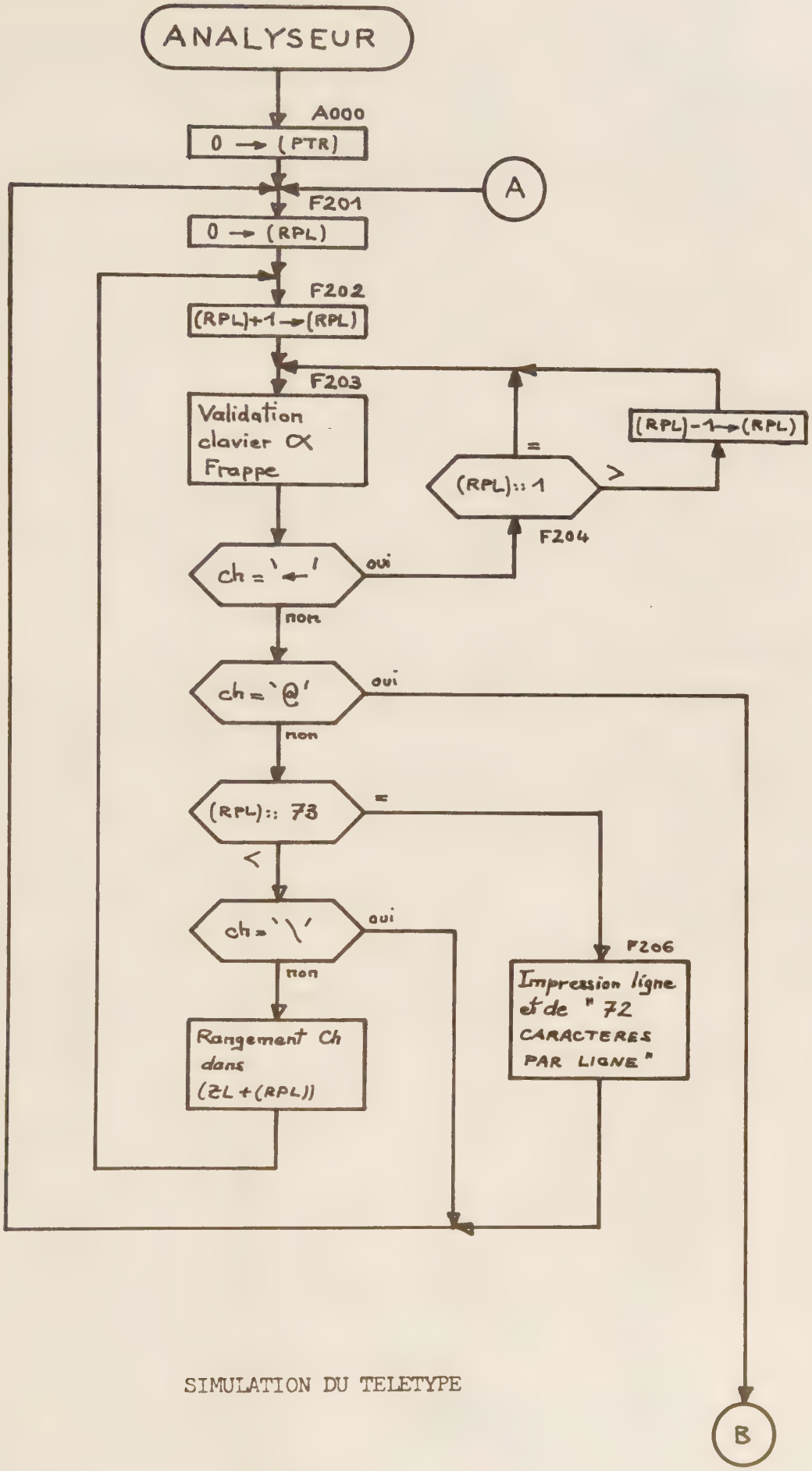


Figure B-1

NB représente le numéro de la piste qui va recevoir le bloc courant quand celui-ci sera plein. PTR est le compteur du nombre d'instructions enregistrées dans le bloc courant.

Au départ, NB et PTR contiennent 0. L'enregistrement normal par numéros croissants se fait par B000, B001, B002, B027 et B004 (figure B-4). En B002, il y a comparaison entre le numéro de la ligne courante et le numéro de la dernière ligne enregistrée. Plaçons-nous dans le cas où le résultat est > 0 (numéros croissants), AAA est à 0 (c'est une nouvelle ligne à enregistrer), il y a progression du pointeur PMC, et de PTR, si PTR est inférieur à 9, le bloc n'est



SIMULATION DU TELETYPE

Figure B-2

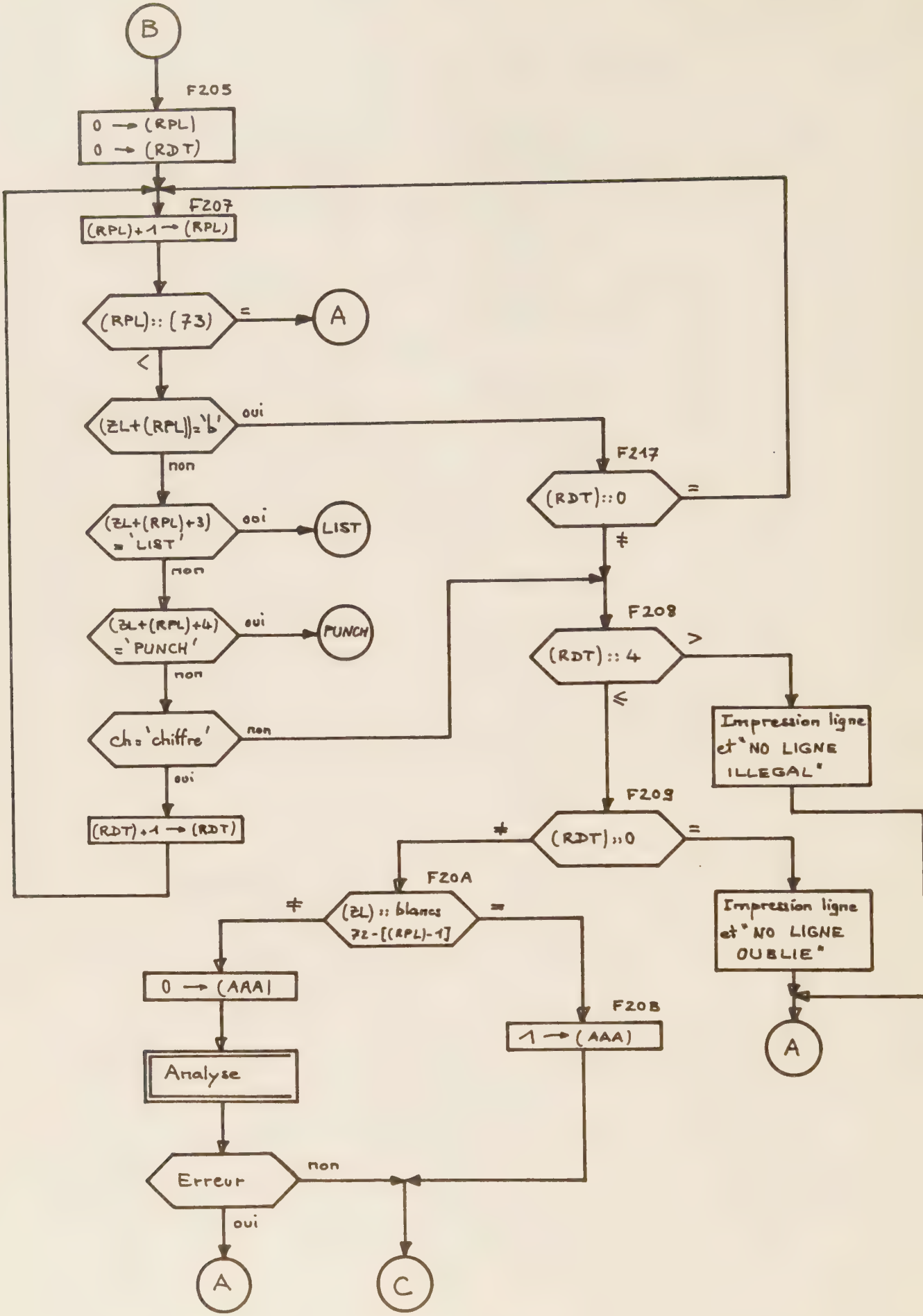

B
F205
0 → (RPL)
0 → (RDT)
F207
(RPL)+1 → (RPL)
(RPL):: (73)
=
A
<
(ZL+(RPL))='b'
oui
non
F217
(RDT)::0
=
≠
(ZL+(RPL)+3) = 'LIST'
oui
LIST
non
(ZL+(RPL)+4) = 'PUNCH'
oui
PUNCH
non
F208
(RDT) :: 4
>
≤
ch='chiffre'
non
oui
Impression ligne et "NO LIGNE ILLEGAL"
(RDT)+1 → (RDT)
F209
(RDT) :: 0
=
≠
F20A
(ZL) :: blancs 72-[(RPL)-1]
≠
=
Impression ligne et "NO LIGNE OUBLIE"
0 → (AAA)
F20B
1 → (AAA)
A
Analyse
Erreur
non
oui
A
C

Figure B-3

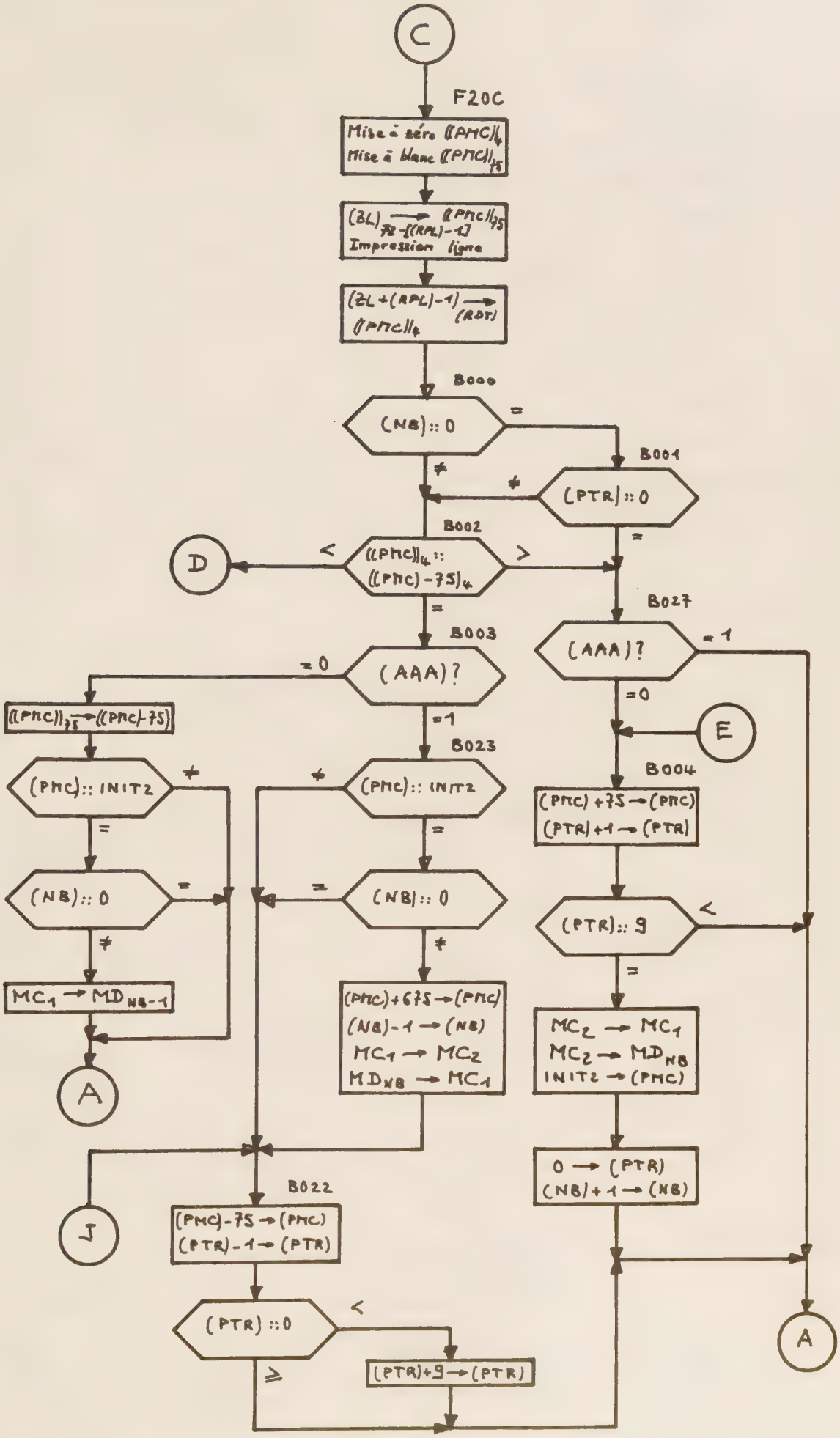

C
F20C
Mise à zéro ((PMC))4
Mise à blanc ((PMC))75
(BL) → ((PMC))75
Impression ligne
(BL)+((RPL))-1 → (RDT)
((PMC))4
B000
(NB) :: 0
=
≠
B001
(PTR) :: 0
≠
=
B002
D
((PMC))4 :: ((PMC)-75)4
<
>
=
B027
(AAA) ?
=1
=0
B003
(AAA) ?
=0
=1
E
((PMC))75 → ((PMC)-75)
B023
B004
(PMC) :: INIT2
(PMC) :: INIT2
(PMC)+75 → (PMC)
(PTR)+1 → (PTR)
≠
=
(NB) :: 0
(NB) :: 0
(PTR) :: 9
<
MC1 → MDNB-1
(PMC)+675 → (PMC)
(NB)-1 → (NB)
MC1 → MC2
MDNB → MC1
MC2 → MC1
MC2 → MDNB
INIT2 → (PMC)
A
J
B022
(PMC)-75 → (PMC)
(PTR)-1 → (PTR)
0 → (PTR)
(NB)+1 → (NB)
(PTR) :: 0
<
≥
(PTR)+9 → (PTR)
A

Figure B-4

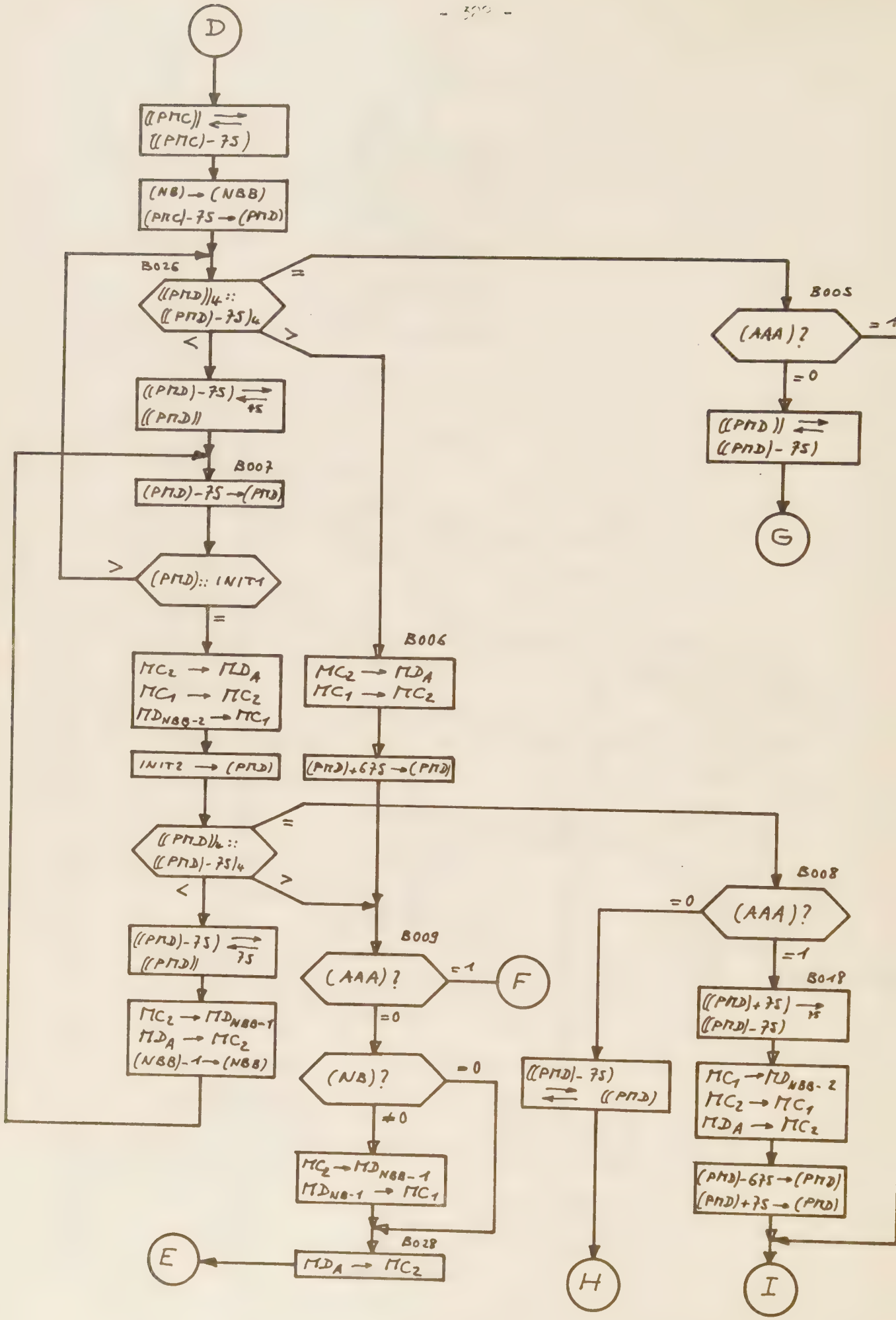

D
((PMC)) ⇄ ((PMC)-75)
(NB) → (NBB)
(PMC)-75 → (PMD)
B026
((PMD))4 :: ((PMD)-75)4
=
>
<
((PMD)-75) ⇄ ((PMD)) ±5
B007
(PMD)-75 → (PMD)
>
(PMD):: INIT1
=
MC2 → MLDA
MC1 → MC2
MDNBB-2 → MC1
INIT2 → (PMD)
B006
MC2 → MDA
MC1 → MC2
(PMD)+675 → (PMD)
((PMD))4 :: ((PMD)-75)4
=
>
<
((PMD)-75) ⇄ ((PMD)) 75
MC2 → MDNBB-1
MDA → MC2
(NBB)-1 → (NBB)
B009
(AAA)?
=1
F
=0
(NB)?
=0
≠0
MC2 → MDNBB-1
MDNB-1 → MC1
B028
MDA → MC2
E
B005
(AAA)?
=1
=0
((PMD)) ⇄ ((PMD)-75)
G
B008
=0
(AAA)?
=1
((PMD)-75) ⇄ ((PMD))
H
B018
((PMD)+75) → ((PMD)-75) 75
MC1 → MDNBB-2
MC2 → MC1
MDA → MC2
(PMD)-675 → (PMD)
(PMD)+75 → (PMD)
I

Figure B-5

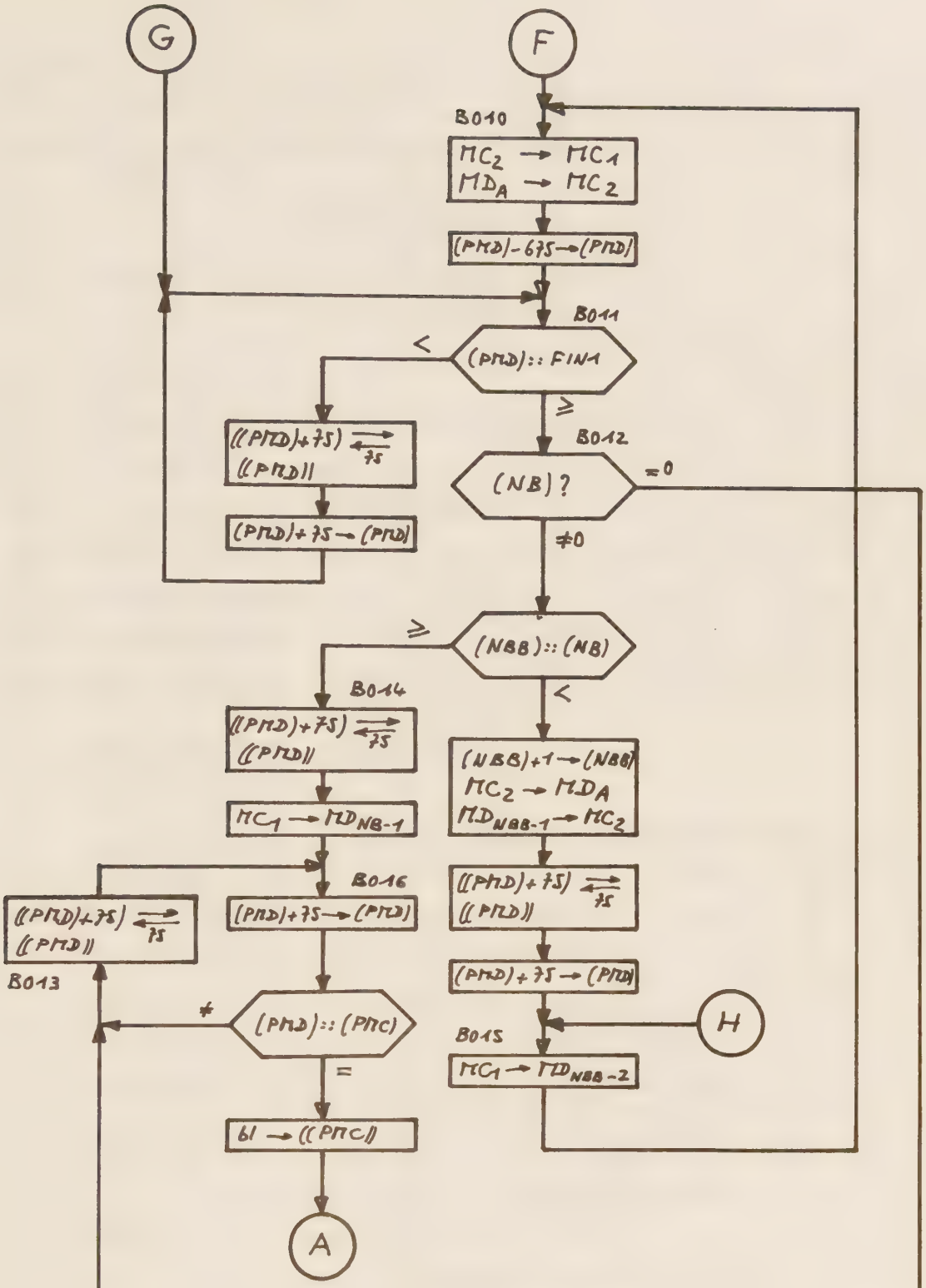

G
F
B010
MC2 → MC1
MDA → MC2
(PMD)-675→(PMD)
B011
(PMD):: FIN1
<
≥
((PMD)+75) ⇄ ((PMD)) 75
(PMD)+75→(PMD)
B012
(NB) ?
=0
≠0
(NBB):: (NB)
≥
<
B014
((PMD)+75) ⇄ ((PMD)) 75
MC1 → MD NB-1
(NBB)+1→(NBB)
MC2 → MDA
MD NBB-1 → MC2
B016
((PMD)+75) ⇄ ((PMD)) 75
(PMD)+75→(PMD)
((PMD)+75) ⇄ ((PMD)) 75
B013
(PMD)+75→(PMD)
≠
(PMD):: (PMC)
=
H
B015
MC1 → MD NBB-2
61 → ((PMC))
A

Figure B-6

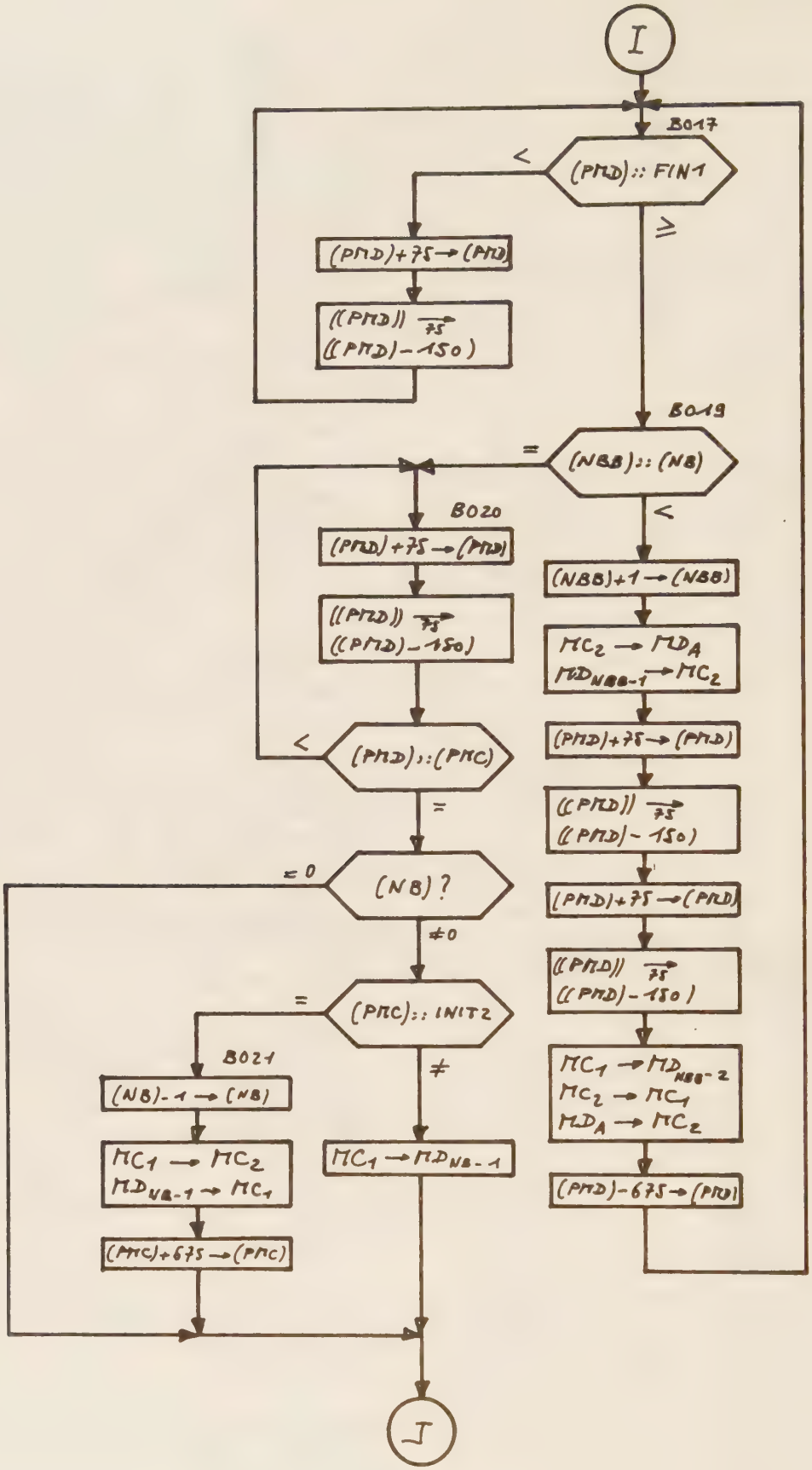

I
B017
(PMD) :: FIN1
<
≥
(PMD)+75 → (PMD)
((PMD)) 75 ((PMD)-150)
B019
(NBB) :: (NB)
=
<
B020
(PMD)+75 → (PMD)
((PMD)) 75 ((PMD)-150)
(NBB)+1 → (NBB)
MC2 → MDA
MDNBB-1 → MC2
(PMD)+75 → (PMD)
((PMD)) 75 ((PMD)-150)
(PMD)+75 → (PMD)
((PMD)) 75 ((PMD)-150)
MC1 → MDNBB-2
MC2 → MC1
MDA → MC2
(PMD)-675 → (PMD)
(PMD) :: (PMC)
<
=
(NB) ?
=0
≠0
(PMC) :: INIT2
=
≠
B021
(NB)-1 → (NB)
MC1 → MC2
MDNB-1 → MC1
(PMC)+675 → (PMC)
MC1 → MDNB-1
J

Figure B-7

pas plein, on retourne à la frappe. Si PTR égale 9, le bloc est plein, il y a recopie de MC2 dans MC1 et enregistrement du bloc dans la piste courante MD_{NB}, puis réinitialisation de PMC et de PTR, enfin progression de NB.

B-4 - MODIFICATION DE LA DERNIERE LIGNE ENREGISTREE

Pour modifier la ligne qui vient d'être enregistrée, on frappe le même numéro de ligne suivie du texte nouveau, le cas échéant, en B002, le résultat du test est =, si AAA égale 1, il convient d'éliminer la ligne par recul du pointeur PMC et régression de PTR, si PMC égale INIT2, il faut, dans le cas où un bloc au moins a déjà été transféré en tambour, récupérer le dernier bloc dans MC2 (il est dans MC1) et remettre MC1 à jour. Si AAA égale 0, on recopie la cellule courante dans la cellule précédente, il faut, le cas échéant, remettre à jour le bloc dernièrement enregistré en tambour.

B-5 - ELIMINATION D'UNE LIGNE ANTERIEURE

Le numéro contenu dans le cellule courante est inférieur à celui de la cellule précédente, c'est-à-dire qu'en B002, le résultat de la comparaison est $<$, il y a branchement en D (figure B-5).

On fait l'échange des deux lignes ((PMC)) et ((PMC)-75), un pointeur de recherche PMD est initialisé sur l'avant-dernière cellule (qui contient maintenant le numéro de la ligne à éliminer). Il y a échanges successifs des lignes avec récupération et remise à jour des blocs en tambour, jusqu'à ce qu'on retrouve la ligne à éliminer, il y a alors 2 endroits possibles où la reconnaissance est effectuée et qui correspondent aux 2 branchement possibles B005 et B008. Dans les 2 cas, on aboutit à I (il y a traitement particulier en B018, qui annule l'effet de la séquence précédant la comparaison qui a permis d'aboutir à B008). Dans I (figure B-7), le traitement s'appuie sur le fait que pour éliminer une ligne, il

suffit de recopier la ligne courante dans celle qui la précède de 2 cellules (car il y a en fait 2 lignes à éliminer, celle qui était enregistrée et la ligne blanche introduite, ceci pour rester compatible avec le traitement du remplacement de ligne).

On fait redéfiler les blocs dans le sens croissant jusqu'à épuisement avec remises à jour successives de chacun d'eux.

B-6 - REMPLACEMENT D'UNE LIGNE ANTERIEURE

Le principe du recul dans les blocs par échange de lignes reste le même que ci-dessus, on arrive également en B005 ou B008, quand la ligne à remplacer est juste au-dessus de la nouvelle ligne. Dans les 2 cas, le remplacement est effectué avant d'aboutir respectivement en G ou H (figure B-6).

Il y a maintenant défilement et mise à jour des blocs.

B-7 - INSERTION D'UNE NOUVELLE LIGNE

Le principe des échanges reste le même, la sortie des tests de comparaison est maintenant $>$, et permet d'aboutir soit en B006, soit en B009. Ici, il n'y a pas besoin de faire défiler les blocs dans le sens croissant, car ils sont correctement remplis, le passage en E correspond au fait qu'on ajoute une ligne à l'ensemble.

B-8 - ELIMINATION D'UNE LIGNE INEXISTANTE

Il faut prévoir l'erreur qui consiste à vouloir éliminer une ligne qui n'existe pas. Il suffit en B009 (figure B-5) d'aller en F (figure B-6) où le traitement s'effectue comme si on avait remplacé une ligne antérieure.

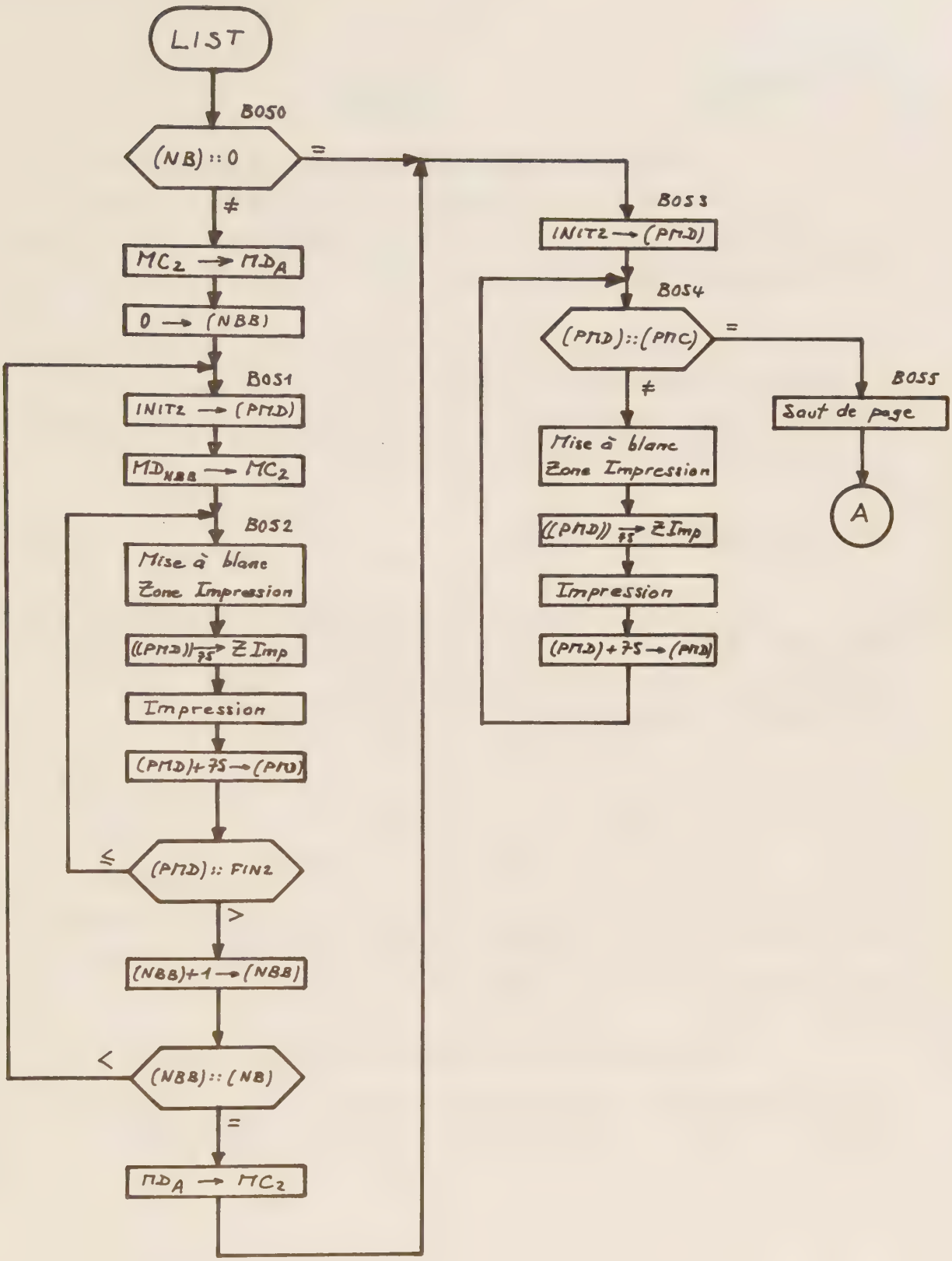

LIST
B050
(NB) :: 0
=
≠
MC2 → MDA
0 → (NBB)
B051
INIT2 → (PMD)
MDNBB → MC2
B052
Mise à blanc Zone Impression
((PMD)) 75 → ZImp
Impression
(PMD)+75 → (PMD)
≤
(PMD) :: FIN2
>
(NBB)+1 → (NBB)
<
(NBB) :: (NB)
=
MDA → MC2
B053
INIT2 → (PMD)
B054
(PMD) :: (PMC)
=
≠
Mise à blanc Zone Impression
((PMD)) 75 → ZImp
Impression
(PMD)+75 → (PMD)
B055
Saut de page
A

Figure B-8

B-9 - REMARQUES

Les organigrammes qu'on vient de décrire sommairement, sont d'une complexité assez grande, mais ils offrent l'avantage que le traitement ne nécessite qu'une piste supplémentaire (MD_A) en dehors de celles normalement utilisées pour enregistrer les blocs.

On peut néanmoins penser qu'une étude approfondie de ces organigrammes permettrait une simplification, en ce sens qu'on pourrait regrouper en un seul, des organigrammes relativement semblables comme I et G par exemple.

B-10 - COMMANDES LIST et PUNCH

On donne ici (figure B-8) l'organigramme de traitement de LIST. Il y a défilement des blocs depuis le premier jusqu'au bloc courant, avec prélèvement de l'image de chaque ligne et transfert dans la zone d'impression.

La piste MD_A sert à sauvegarder le bloc courant de MC2.

Le listage se termine par un saut de page. Remarquons qu'il est possible de faire autant de listages que l'on veut pendant l'introduction du programme BASIC. (C'est le même algorithme qui est utilisé pour le listage dans l'interpréteur).

L'organigramme de traitement de PUNCH est le même, il y a remplacement des commandes d'impression par des commandes de perforation et l'achèvement se fait par une fin de travail (End of Job).

ANNEXE C

LISTE DES ERREURS DE SYNTAXE

Erreur n° 1 Signe de division dans une opération matricielle.

2 Présence d'un signe de relation dans une instruction autre que IF.

3 Mot-clé inconnu.

4 Dans une instruction DEF, il manque la variable formelle.

5 Dans une instruction de définition de fonction, il manque la parenthèse d'ouverture.

6 Dans une instruction de définition de fonction, présence de symboles séparant FNα de (β).

7 Dans une instruction de définition de fonction, présence de symboles entre le paramètre formel et la parenthèse de fermeture, ou bien la parenthèse d'ouverture est remplacée par un autre délimiteur.

8 * Dans une instruction INPUT/READ, la liste de variables est incorrecte.

9 * Hors liste.

10 Il manque la parenthèse de fermeture dans une expression entre parenthèses .

11 * Hors liste .

12 * Erreur inhibée par l'erreur 42 .

13 * Erreur inhibée par l'erreur 43 .

14 * Erreur inhibée par l'erreur 42 .

15 Dans une instruction du type MATREAD, MATINPUT, MAT α = ZER/CON/IDN, la parenthèse de fermeture est omise ou remplacée par un délimiteur invalide .

16 Il y a plus de 2 dimensions dans un redimensionnement .

17 Dans une instruction matricielle du type MAT α = ZER/CON/IDN, l'instruction est incorrecte entre MAT et ZER/CON/IDN .

18 * Dans une structure de nombre, la partie exposant n'est pas précédée par un nombre décimal ou par un entier pointé .

19 * Erreur impossible .

20 Dans une instruction REM, le mot-clé est suivi par des guillemets .

21 * Erreur impossible .

22 Il y a plus de 2 indices dans une variable indicée .

23 Il manque une parenthèse de fermeture dans une structure de variable indicée .

24 Dans une instruction DIM, un tableau est défini avec plus de 2 dimensions .

25 Dans une instruction DIM, il manque une parenthèse de fermeture dans un dimensionnement .

26 Dans une instruction DATA, il manque une virgule entre deux constantes .

27 * Erreur impossible .

28 * Erreur impossible .

29 Dans les instructions comportant un numéro de ligne, ce dernier est précédé par un symbole illégal .

30 Dans une instruction IF...THEN, il y a des symboles interposés entre IF et la relation .

31 Dans un ordre de multiplication d'une matrice par une expression, le symbole '*' est remplacé par '+' ou '-'.

32 Dans une instruction DIM, il manque une virgule entre 2 tableaux .

33 * Erreur impossible .

34 Dans une instruction matricielle autre que MATINPUT/ MATREAD, il y a une virgule derrière un tableau .

35 * Un tableau est utilisé hors d'un contexte MATINPUT ou MATREAD .

36 Dans une instruction matricielle du type MAT, il manque le symbole d'affectation matricielle .

37 Dans une instruction matricielle du type MAT α = INV/TRN(β), il manque le symbole d'affectation matricielle .

38 Il y a un mot-clé précédant une instruction valide .

39 Il y a un délimiteur invalide derrière une instruction valide .

40 Dans une instruction DEF, il y a un mot-clé invalide après le mot-clé DEF .

41 Dans une instruction IF, il y a emploi de TØ ou STEP à la place de THEN .

42 Dans une instruction FØR, il y a emploi de THEN ou STEP à la place de TØ .

43 Dans une instruction FØR, il y a emploi de THEN ou TØ à la place de STEP .

44 Dans une instruction PRINT, il manque une virgule ou un point-virgule entre 2 chaînes de caractères entre guillemets .

45 Il manque une parenthèse de fermeture derrière INV(α ou TRN(α .

46 Dans une instruction IF, 2 signes de relation se suivent .

47 Deux symboles d'opération se succèdent .

48* Erreur impossible .

49* Une liste de variables n'est pas précédée par un des mots-clés INPUT ou READ .

50 Il y a 2 délimiteurs qui se succèdent illégalement, ou un symbole d'opération à un endroit inconvenable .

51 Il y a affectation multiple dans une instruction autre que LET .
Dans une instruction FØR...TØ...STEP, il y a des délimiteurs interposés entre la variable de boucle et le signe égal .

52 Dans une instruction PRINT, il y a un délimiteur supplémentaire à droite .

61 La variable citée dans l'ordre NEXT ne fait pas référence à l'ordre FØR précédent (boucles imbriquées) .

62 Le dernier ordre du programme n'est pas un END .

63 Il manque au moins un ordre NEXT .

Un certain nombre d'erreurs sont accompagnées d'une marque '*'. On peut classer ces erreurs en 4 catégories :

- erreurs ne faisant pas partie du répertoire d'erreurs. Leurs numéros sont libres.

- erreurs inhibées par d'autres très précises. Au cours de la mise au point de l'analyseur, les traitements étant modifiés, certaines erreurs très précises ont été détectées plus en avant, il s'agit en fait de changements de numéros d'erreur.

- erreurs impossibles. Mêmes origines que ci-dessus, ce sont des erreurs non précises détectées plus en avant.

- autres. Ces erreurs sont hautement improbables, le diagnoctic donné est théorique, car en pratique elles ne sont pas apparues dans les essais.

ANNEXE D

EXEMPLES D'ARBRES D'ANALYSE SYNTAXIQUE

D-1 - INTRODUCTION - REMARQUES GENERALES

Les exemples que nous exposons ci-après sont fournis par le programme ANARB (Analyse Syntaxique avec Arbre).En utilisation normale, nous n'employons que le programme ANALY (Analyse Syntaxique sans Arbre).

Le BASIC est un langage suffisamment simple pour qu'une erreur de syntaxe soit identifiée facilement par l'utilisateur.

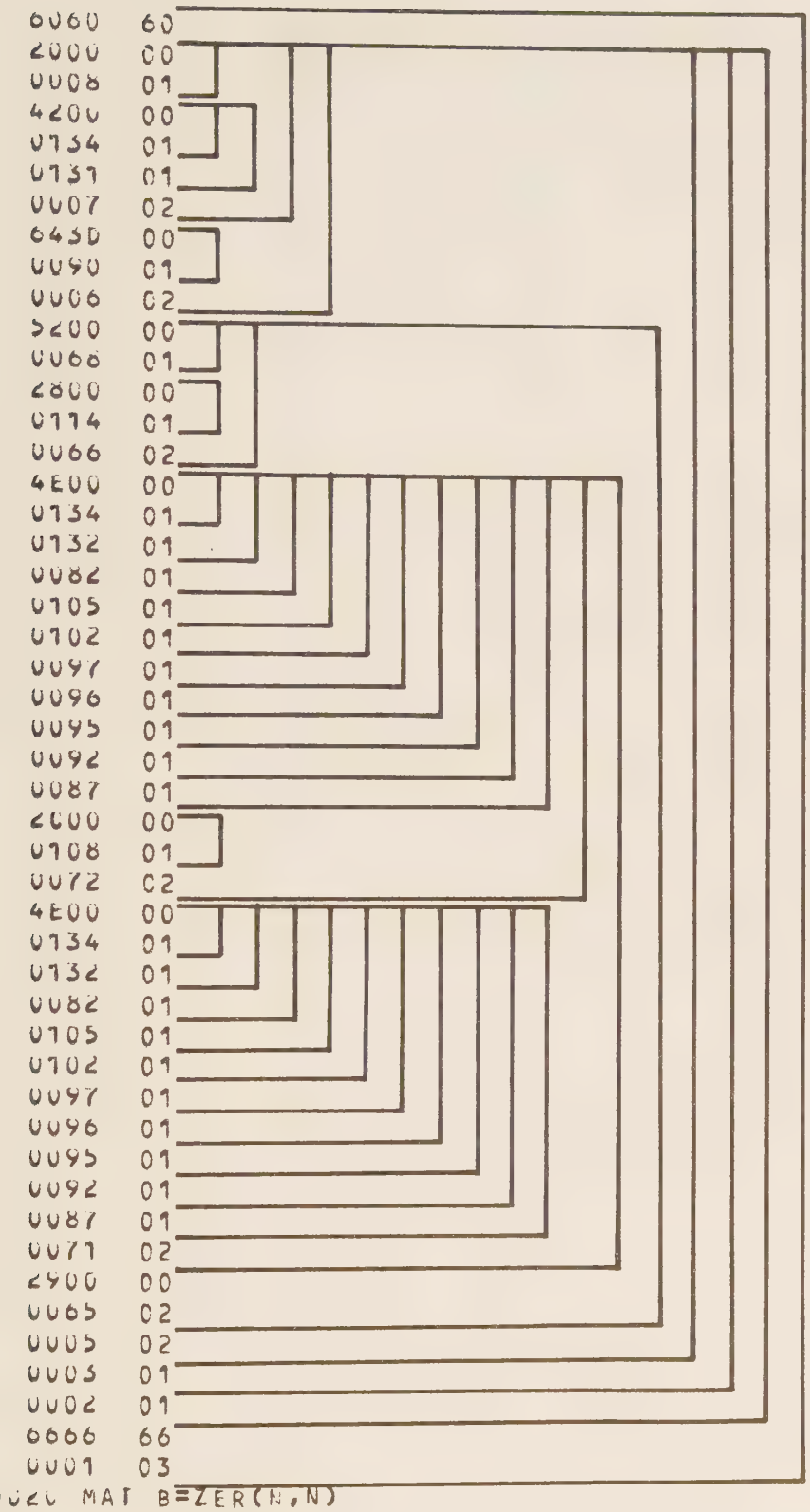
En fait, l'arbre d'analyse syntaxique est surtout utile pour la mise au point de l'analyseur. Cela permet de vérifier la concordance entre l'arbre fourni par l'analyseur et l'arbre qu'on peut écrire à la main en faisant soi-même l'analyse syntaxique.

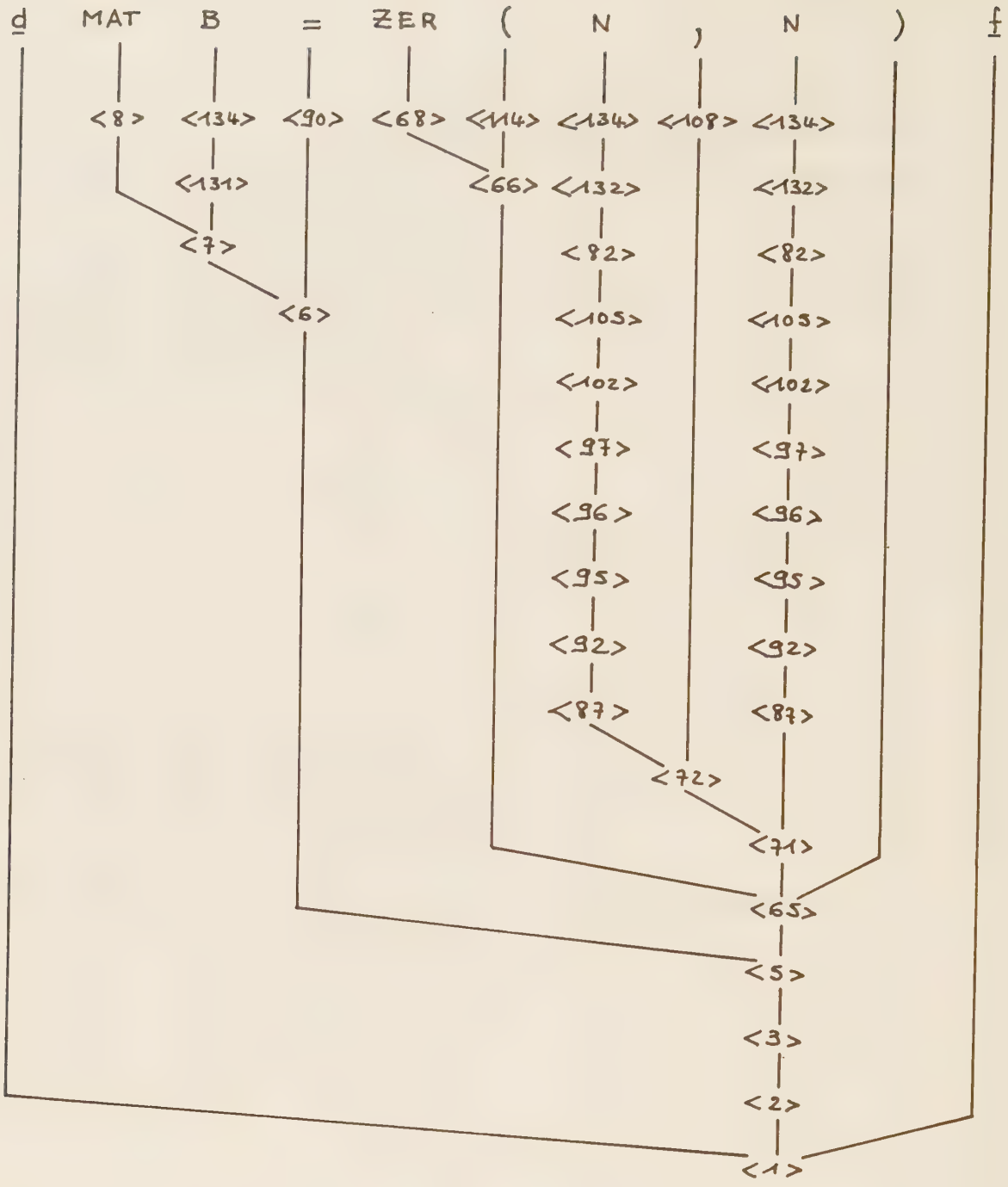
Néanmoins, nous donnons dans les exemples, 2 arbres interrompus par la détection d'une erreur de syntaxe. Le premier exemple montre l'utilisation d'un symbole d'opération interdit, ce qui ne remet pas en cause l'analyse précédente. Dans le second exemple, il y a 2 mots-clés, seul le second entre dans l'analyse, et à la fin on s'aperçoit qu'il y a un code syntaxique qui sépare 'd' du code<2> , ce qui est généralement typique d'un mot-clé dédoublé.

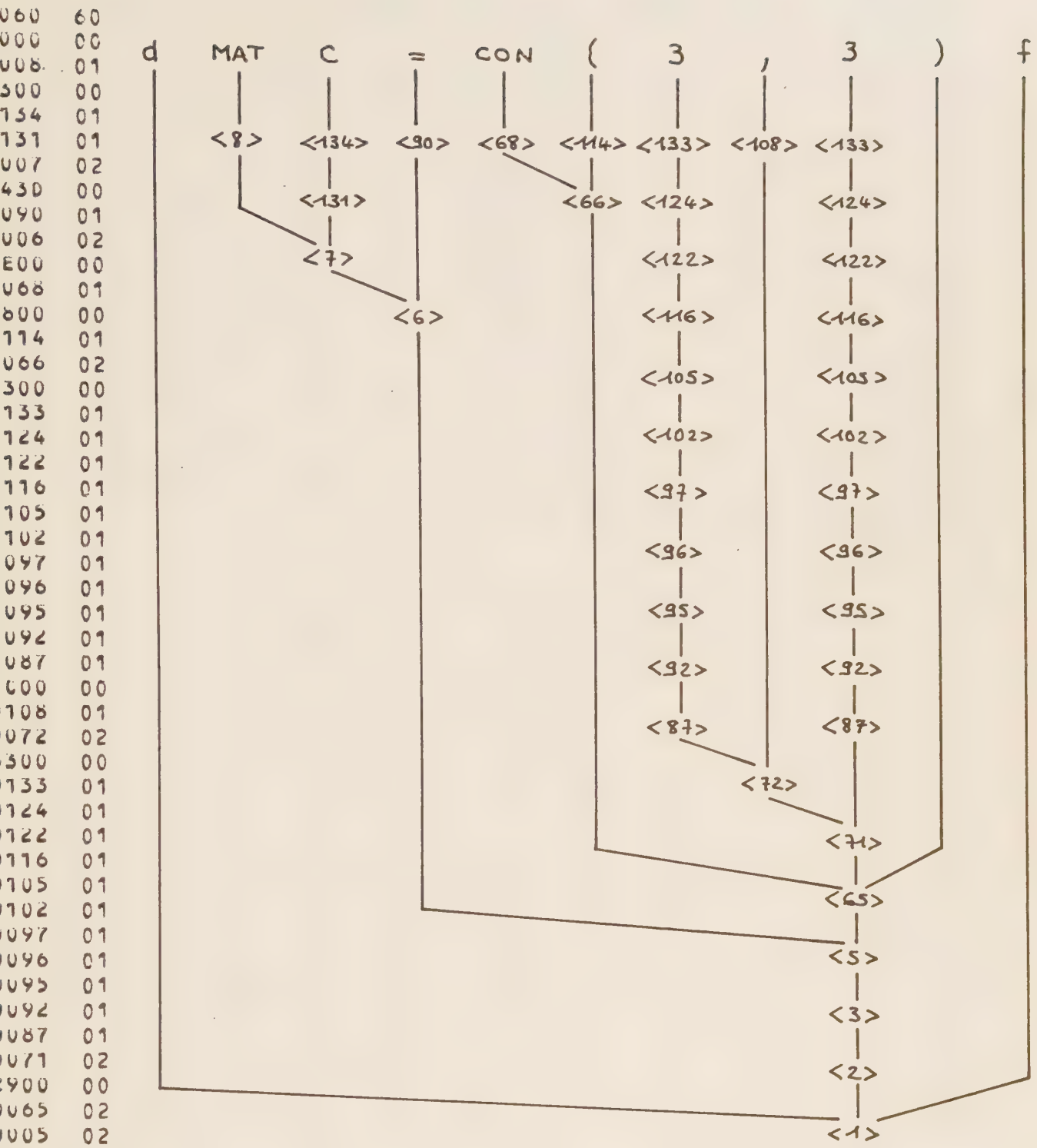
D-2 - EXEMPLES

```
0010    REM QUELQUES EXEMPLES D'ARBRE D'ANALYSE SYNTAXIQUE
0020    MAT B=ZER(N,N)
0030    MAT C=CON(3,3)
0040    MAT X=INV(Y)
0050    MAT A=B*C
0060    MAT S=(SQR(COS(A(1,I)↑2)-1))*Q
0070    IF X<>0 THEN 20
0080    STOP
```

```
0100    REM TRIGONOMETRIE
0110    LET P=3,14159265
0120    DEF FND(X)=X*P/180
0130    DEF FNC(X)=COS(FND(X))
0140    PRINT "COTE","ANGLE","COTE","3EME COTE"
0150    READ X,A,Y
0160    LET Z=SQR(X*X+Y*Y-2*X*Y*FNC(A))
0170    PRINT X,A,Y,Z
0180    GOTO 150
0190    DATA 10,60,20
0200    DATA 3,90,4
0210    DATA 10,120,10
0220    DATA 5,60,5
9999    END
```

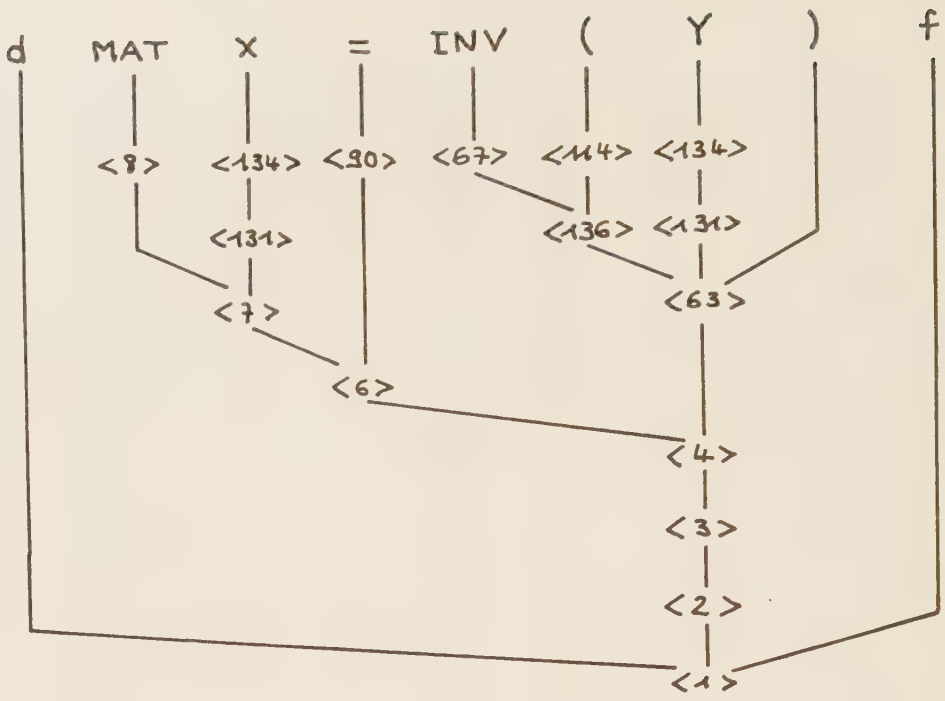

0060 60
2000 00
0008 01
4200 00
0134 01
0131 01
0007 02
6450 00
0090 01
0006 02
5200 00
0066 01
2800 00
0114 01
0066 02
4E00 00
0134 01
0132 01
0082 01
0105 01
0102 01
0097 01
0096 01
0095 01
0092 01
0087 01
2000 00
0108 01
0072 02
4E00 00
0134 01
0132 01
0082 01
0105 01
0102 01
0097 01
0096 01
0095 01
0092 01
0087 01
0071 02
2900 00
0065 02
0005 02
0003 01
0002 01
6666 66
0001 03
020 MAT B=ZER(N,N)


d
MAT
B
=
ZER
(
N
,
N
)
f
<8>
<134>
<90>
<68>
<114>
<134>
<108>
<134>
<131>
<66>
<132>
<132>
<7>
<82>
<82>
<6>
<105>
<105>
<102>
<102>
<97>
<97>
<96>
<96>
<95>
<95>
<92>
<92>
<87>
<87>
<72>
<71>
<65>
<5>
<3>
<2>
<1>

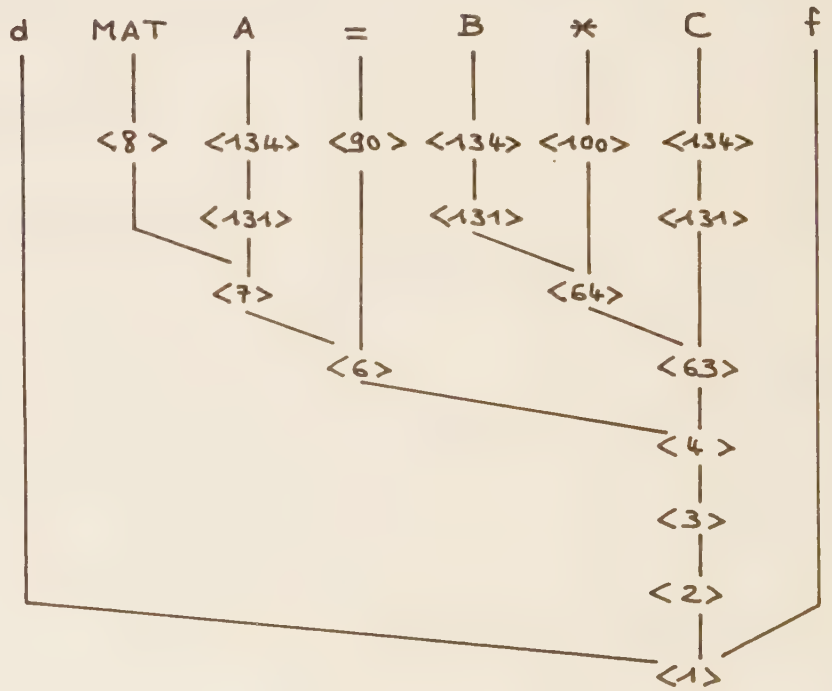



```
060  60
000  00
008  01
500  00
154  01
131  01
007  02
430  00
090  01
006  02
E00  00
068  01
600  00
114  01
066  02
300  00
133  01
124  01
122  01
116  01
105  01
102  01
097  01
096  01
095  01
092  01
087  01
C00  00
108  01
072  02
300  00
133  01
124  01
122  01
116  01
105  01
102  01
097  01
096  01
095  01
092  01
087  01
071  02
900  00
065  02
005  02
003  01
002  01
666  66
001  03
60 MAT C=CON(3,3)
```

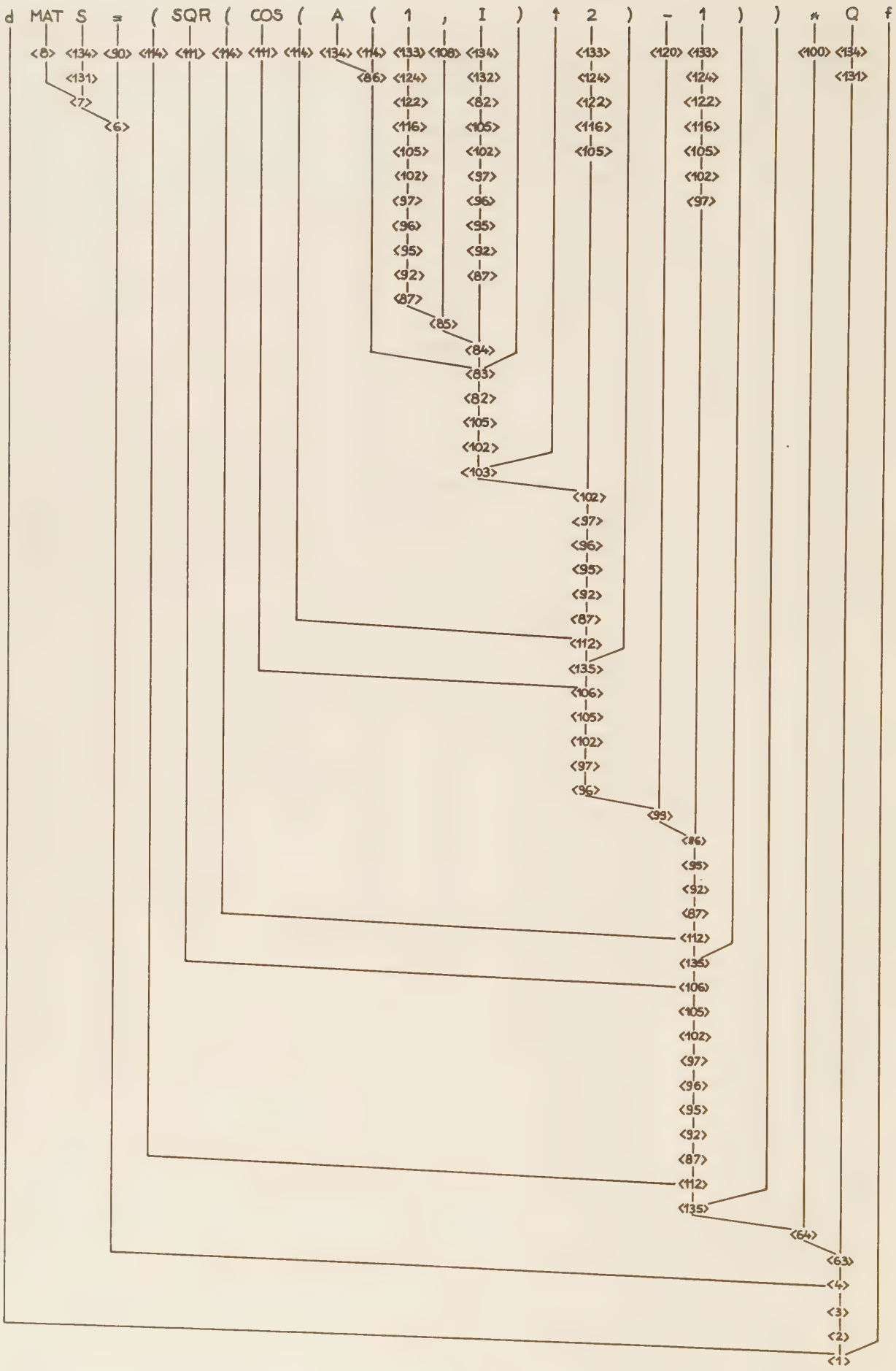
```
0060  60
2000  00
0008  01
5800  00
0134  01
0131  01
0007  02
643D  00
0090  01
0006  02
5600  00
0067  01
2800  00
0114  01
0136  02
5900  00
0134  01
0131  01
2900  00
0163  02
0004  02
0003  01
0002  01
6666  66
0001  03
0040 MAT X=INV(Y)
```



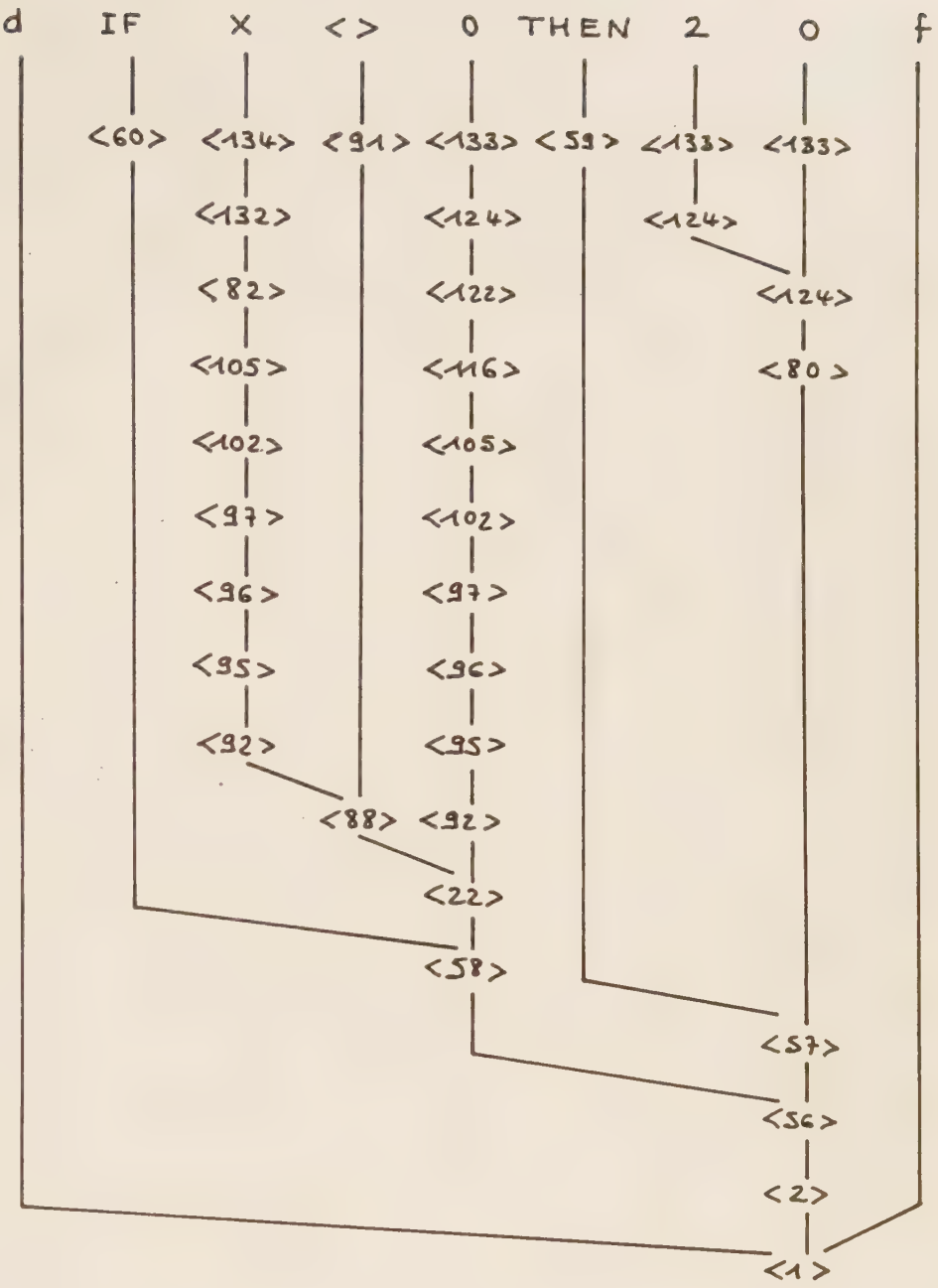

```
6060  60
2000  00
0006  01
4100  00
0134  01
0131  01
0007  02
643D  00
0090  01
0006  02
4200  00
0134  01
0131  01
6100  00
0100  01
0064  02
4300  00
0134  01
0131  01
0063  02
0004  02
0003  01
0002  01
6666  66
0001  03
0050 MAT A=B*C
```



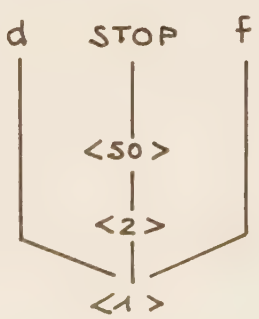
```
0060  60        0087  01        0097  01
2000  00        0084  02        0096  01
0008  01        2900  00        0095  01
5300  00        0083  03        0092  01
0134  01        0082  01        0087  01
0131  01        0105  01        0112  02
0007  02        0102  01        2900  00
643D  00        5E00  00        0155  02
0090  01        0103  02        6100  00
0006  02        5200  00        0100  01
2800  00        0133  01        0064  02
0114  01        0124  01        5100  00
5200  00        0122  01        0134  01
0111  01        0116  01        0131  01
2800  00        0105  01        0065  02
0114  01        0102  02        0004  02
5300  00        0097  01        0003  01
0111  01        0096  01        0002  01
2800  00        0095  01        6666  66
0114  01        0092  01        0001  03
4100  00        0087  01    0060 MAT S=(SQR(COS(A(1,I)*2)-1))*Q
0134  01        0112  02
2800  00        2900  00
0114  01        0155  02
0086  02        0106  02
5100  00        0105  01
0133  01        0102  01
0124  01        0097  01
0122  01        0096  01
0116  01        2000  00
0105  01        0120  01
0102  01        0099  02
0097  01        5100  00
0096  01        0133  01
0095  01        0124  01
0092  01        0122  01
0087  01        0116  01
2600  00        0105  01
0108  01        0102  01
0085  02        0097  01
4900  00        0096  02
0134  01        0095  01
0132  01        0092  01
0082  01        0087  01
0105  01        0112  02
0102  01        2900  00
0097  01        0155  02
0096  01        0106  02
0095  01        0105  01
0092  01        0102  01
```

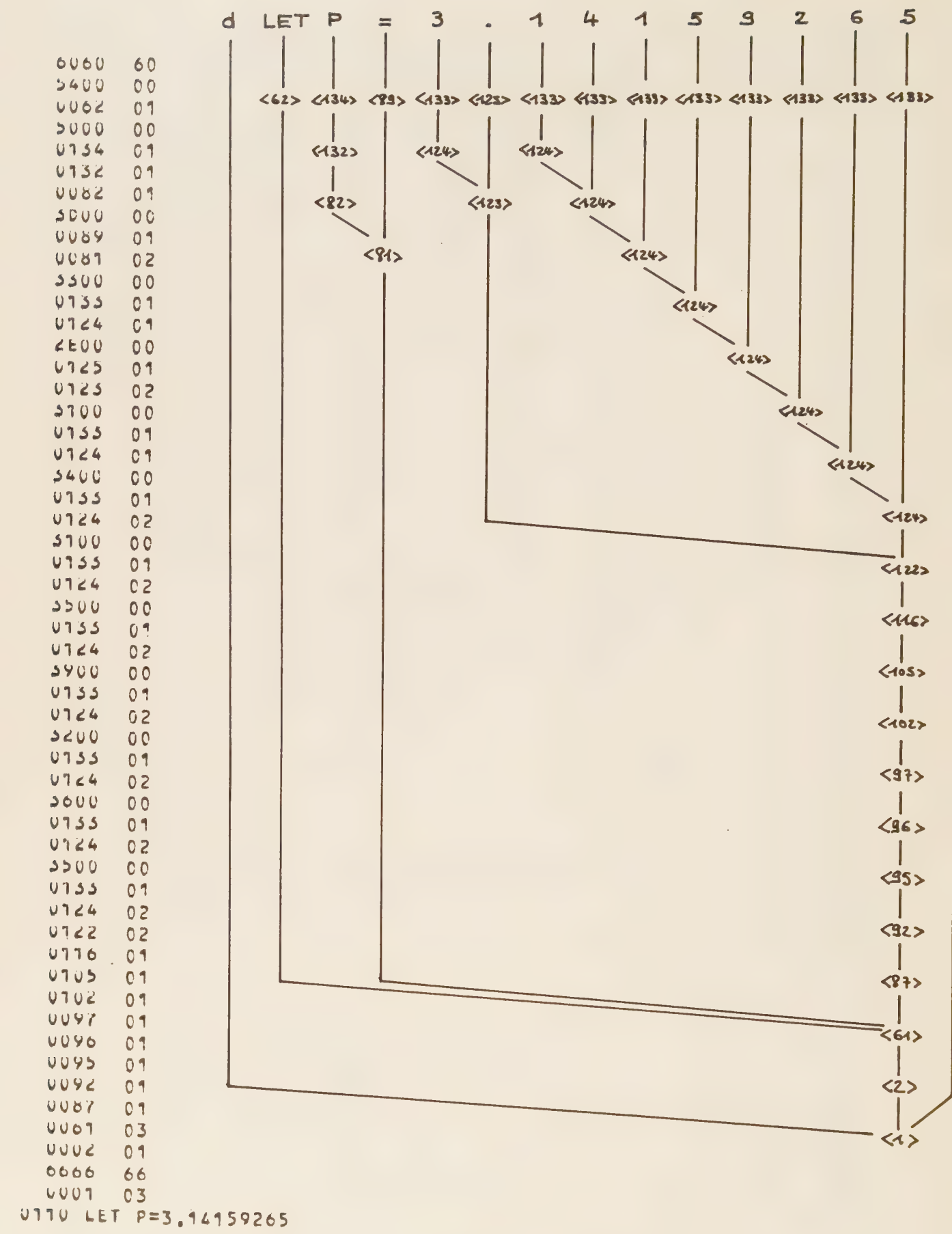

d MAT S = (SQR (COS (A (1 , I) ↑ 2) - 1)) * Q f
<8> <134> <90> <114> <111> <114> <111> <114> <134> <114> <133> <108> <134> <133> <120> <133> <100> <134>
<131> <124> <132> <124> <124>
<7> <122> <82> <122> <122>
<6> <116> <105> <116> <116>
<105> <102> <105> <105>
<102> <97> <102>
<97> <96> <97>
<96> <95>
<95> <92>
<92> <87>
<87>
<85>
<84>
<83>
<82>
<105>
<102>
<103>
<102>
<97>
<96>
<95>
<92>
<87>
<112>
<135>
<106>
<105>
<102>
<97>
<96>
<99>
<86>
<95>
<92>
<87>
<112>
<135>
<106>
<105>
<102>
<97>
<96>
<95>
<92>
<87>
<112>
<135>
<64>
<63>
<4>
<3>
<2>
<1>

```
0060  60
4600  00
0060  01
5800  00
0134  01
0132  01
0082  01
0105  01
0102  01
0097  01
0096  01
0095  01
0092  01
6500  00
0091  01
0088  02
5000  00
0133  01
0124  01
0122  01
0116  01
0105  01
0102  01
0097  01
0096  01
0095  01
0092  01
0022  02
0058  02
4E00  00
0059  01
5200  00
0133  01
0124  01
5000  00
0133  01
0124  02
0080  01
0057  02
0056  02
0002  01
0666  66
0001  03
0070 IF X<>0 THEN 20
```

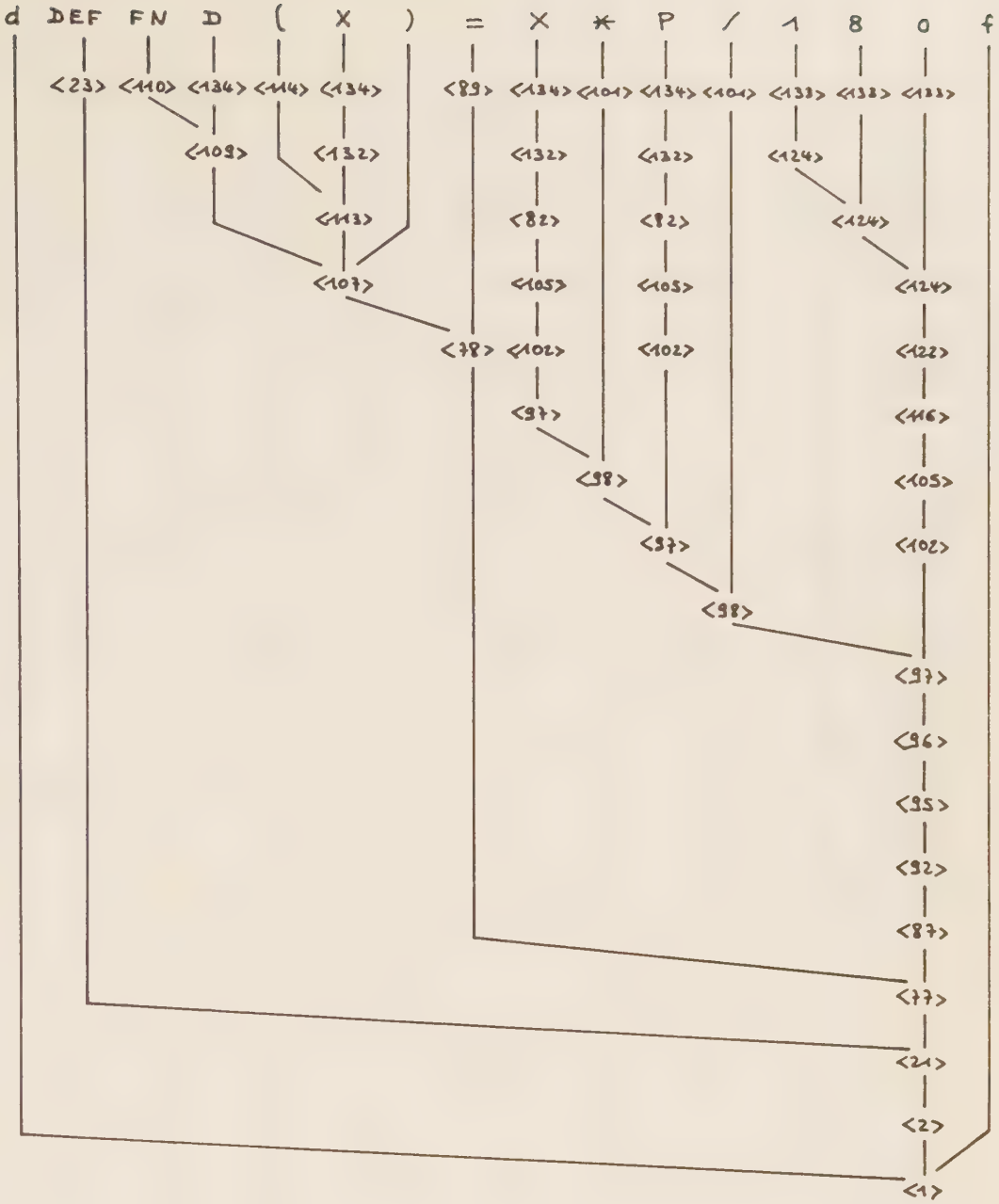



```
0060  60
5000  00
0050  01
0002  01
0666  66
0001  03
0080 STOP
```



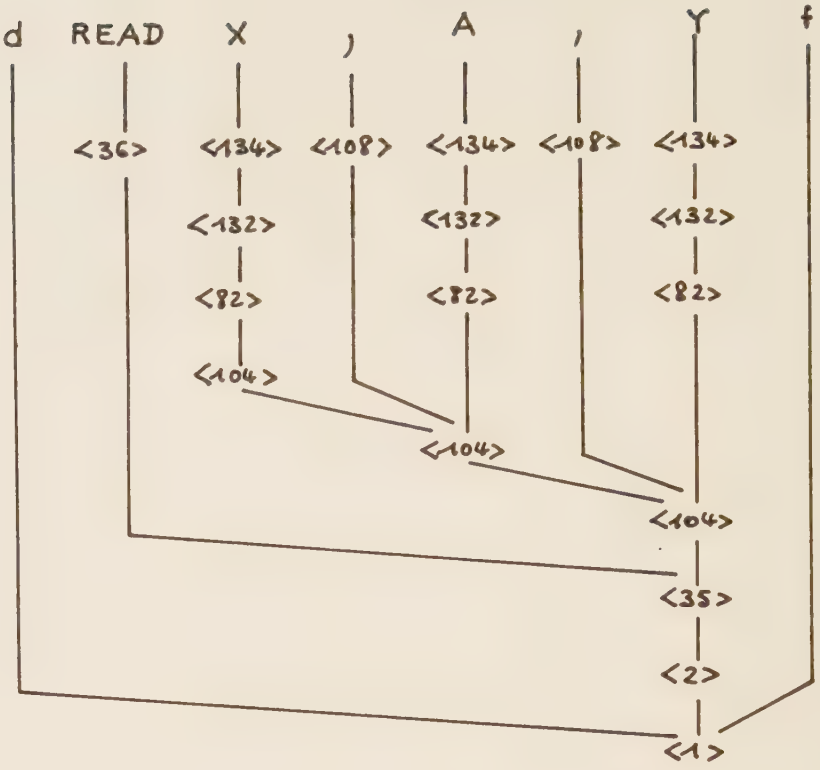

d LET P = 3 . 1 4 1 5 9 2 6 5
0060 60
5400 00
0062 01
5000 00
0134 01
0132 01
0082 01
5000 00
0089 01
0081 02
5500 00
0133 01
0124 01
4E00 00
0125 01
0123 02
5700 00
0133 01
0124 01
5400 00
0133 01
0124 02
5700 00
0133 01
0124 02
5500 00
0133 01
0124 02
5900 00
0133 01
0124 02
5200 00
0133 01
0124 02
5600 00
0133 01
0124 02
5500 00
0133 01
0124 02
0122 02
0116 01
0105 01
0102 01
0097 01
0096 01
0095 01
0092 01
0087 01
0061 03
0002 01
0666 66
0001 03
<62> <134> <89> <133> <123> <133> <133> <131> <133> <133> <133> <133> <133>
<132> <124> <124>
<82> <123> <124>
<81> <124>
<124>
<124>
<124>
<124>
<124>
<124>
<122>
<116>
<105>
<102>
<97>
<96>
<95>
<92>
<87>
<61>
<2>
<1>
0110 LET P=3,14159265

```
0060  60
4000  00
0023  01
404E  00
0110  01
4400  00
0134  01
0109  02
2800  00
0114  01
3800  00
0134  01
0132  01
0113  02
2400  00
0107  02
3000  00
0089  01
0078  02
3800  00
0134  01
0132  01
0082  01
0105  01
0102  01
0097  01
2A00  00
0101  01
0098  02
3000  00
0134  01
0132  01
0082  01
0105  01
0102  01
0097  02
2F00  00
0101  01
0098  02
3100  00
0133  01
0124  01
3800  00
0133  01
0124  02
3000  00
```

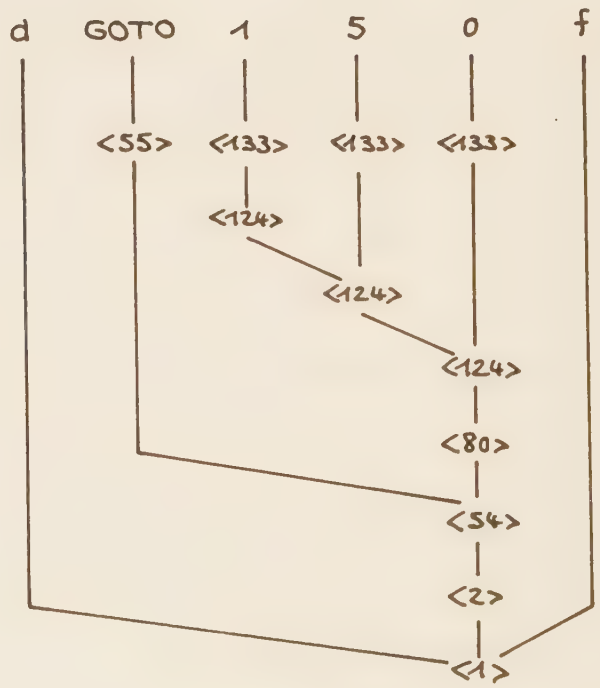


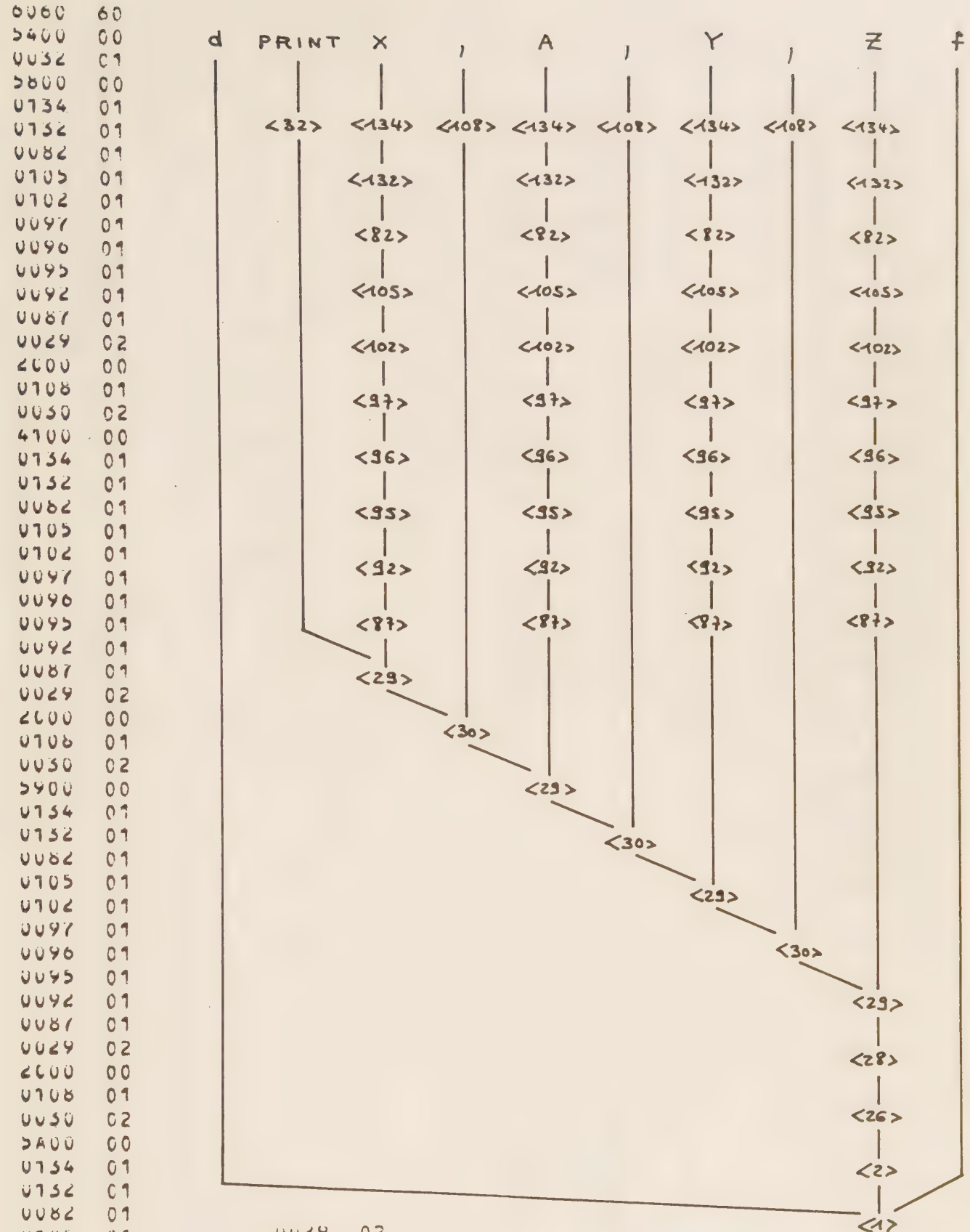

```
0133  01
0124  02
0122  01
0116  01
0105  01
0102  01
0097  02
0096  01
0095  01
0092  01
0087  01
0077  02
0021  03
0002  01
0006  66
0001  03
0120 DEF FND(X)=X*P/180
```

```
0000  60
4400  00
0056  01
5600  00
0154  01
0152  01
0082  01
0104  01
2000  00
0108  01
4100  00
0154  01
0152  01
0082  01
0104  03
2000  00
0108  01
5900  00
0134  01
0132  01
0082  01
0104  03
0055  02
0002  01
0066  66
0001  03
0150 READ X,A,Y
```




```
0000  60
4F00  00
0055  01
5700  00
0153  01
0124  01
5500  00
0153  01
0124  02
5000  00
0153  01
0124  02
0080  01
0054  02
0002  01
0066  66
0001  03
0150 GOTO 150
```

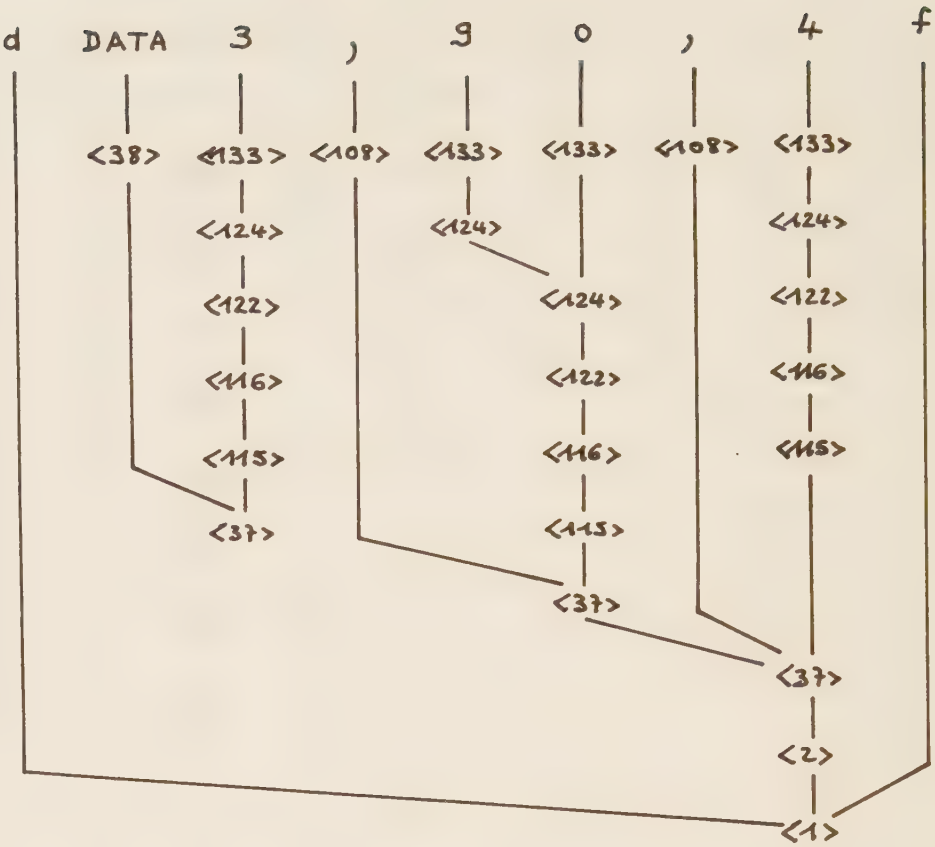




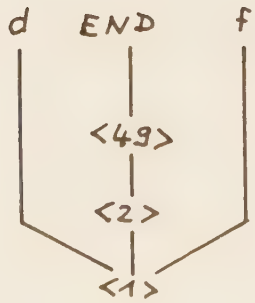

```
0060  60
5400  00
0032  01
5800  00
0134  01
0132  01
0082  01
0105  01
0102  01
0097  01
0096  01
0095  01
0092  01
0087  01
0029  02
2600  00
0108  01
0030  02
4100  00
0134  01
0132  01
0082  01
0105  01
0102  01
0097  01
0096  01
0095  01
0092  01
0087  01
0029  02
2600  00
0108  01
0030  02
5900  00
0134  01
0132  01
0082  01
0105  01
0102  01
0097  01
0096  01
0095  01
0092  01
0087  01
0029  02
2600  00
0108  01
0030  02
5A00  00
0134  01
0132  01
0082  01
0105  01
0102  01
0097  01
0096  01
0095  01
0092  01
0087  01
```

```
0029  02
0028  01
0026  01
0002  01
0066  66
0001  03
0170 PRINT X,A,Y,Z
```

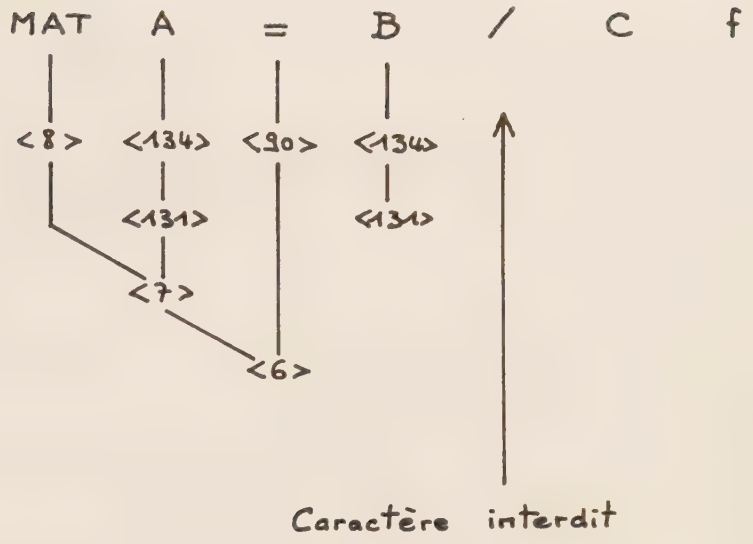
```
0060  60
4100  00
0038  01
3300  00
0133  01
0124  01
0122  01
0116  01
0115  01
0037  02
2000  00
0108  01
3900  00
0133  01
0124  01
3000  00
0133  01
0124  02
0122  01
0116  01
0115  01
0037  03
2000  00
0108  01
3400  00
0133  01
0124  01
0122  01
0116  01
0115  01
0037  03
0002  01
0000  66
0001  03
0200 DATA 3,90,4
```



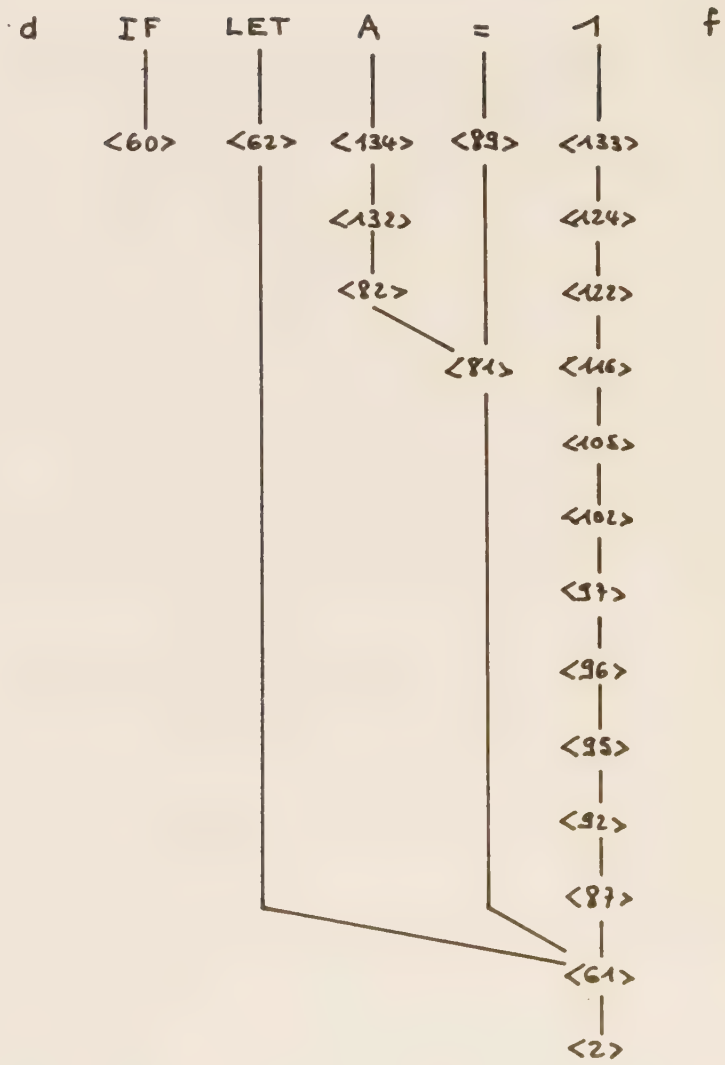

```
0060  60
4400  00
0049  01
0002  01
0000  66
0001  03
9999 END
```



```
   ERREUR NO    1
 6060   60
 2000   00
 0008   01
 4100   00
 0134   01
 0131   01
 0007   02
 643D   00
 0090   01
 0006   02
 4200   00
 0134   01
 0131   01
0010 MAT A=B/C
```




```
   ERREUR NO   38
 6060   60
 4600   00
 0060   01
 5400   00
 0062   01
 4100   00
 0134   01
 0132   01
 0082   01
 5D00   00
 0089   01
 0081   02
 5100   00
 0133   01
 0124   01
 0122   01
 0116   01
 0105   01
 0102   01
 0097   01
 0096   01
 0095   01
 0092   01
 0087   01
 0061   03
 0002   01
1 IF LET A=1
```



ANNEXE E

RECHERCHE DU CARACTERE ENTIER DE L'EXPOSANT POUR A↑X

E-1 - INTRODUCTION

Nous avons vu au § VI-8-3 que l'amélioration du sous-programme d'exponentiation passe par la connaissance du caractère entier ou non-entier, pair ou impair de l'exposant. Nous allons exposer ici la façon de déterminer ces caractères.

E-2 - CARACTERE ENTIER

Pour déterminer si l'exposant est entier, on utilise la propriété que la représentation flottante d'une quantité numérique est décimale.

Rappelons que la quantité est mise sous la forme :

$$0,d_1d_2d_3d_4d_5d_6d_7 \ * \ 10^{E1E2}$$

Figure E-1

où d_1 à d_7 représentent les 7 chiffres significatifs de la quantité et E1, E2 sont les chiffres de l'exposant.

Nous savons également qu'il y a une modification de 50 dans la représentation interne :

$$e_1e_2 = E1E2 + 50$$

En machine, on a :

e_1e_2	0 $d_1d_2d_3d_4d_5d_6d_7$

Figure E-2

(les quantités négatives voient la figure complémentée à 10^8)

Sous la forme externe (figure E-1), si l'on effectue autant de décalages à droite de la virgule qu'il y a d'unités dans E1E2 (à condition que E1E2 soit positif), on détermine facilement si la quantité est entière. Il faut simplement que les chiffres qui restent à droite de la virgule soient des zéros.

En machine, on procède à l'envers, la virgule reste fixe et on décale tous les chiffres vers la gauche. Pour simplifier les comparaisons, on élimine du registre de test, à la fois l'exposant et le chiffre de signe (non significatif) par un décalage de 3 crans à gauche, en fait on obtient le nombre de décalages à effectuer en retranchant 50 - 3 = 47 à l'exposant. Pour éviter un nombre de décalages important, dû à un grand exposant, on vérifie que l'exposant modifié n'est pas supérieur à 10 (7 chiffres + 3 décalages de normalisation), sinon on conclut de suite que l'exposant est entier. Une quantité purement fractionnaire voit son exposant en machine inférieur à 51.

On obtient alors l'organigramme suivant :

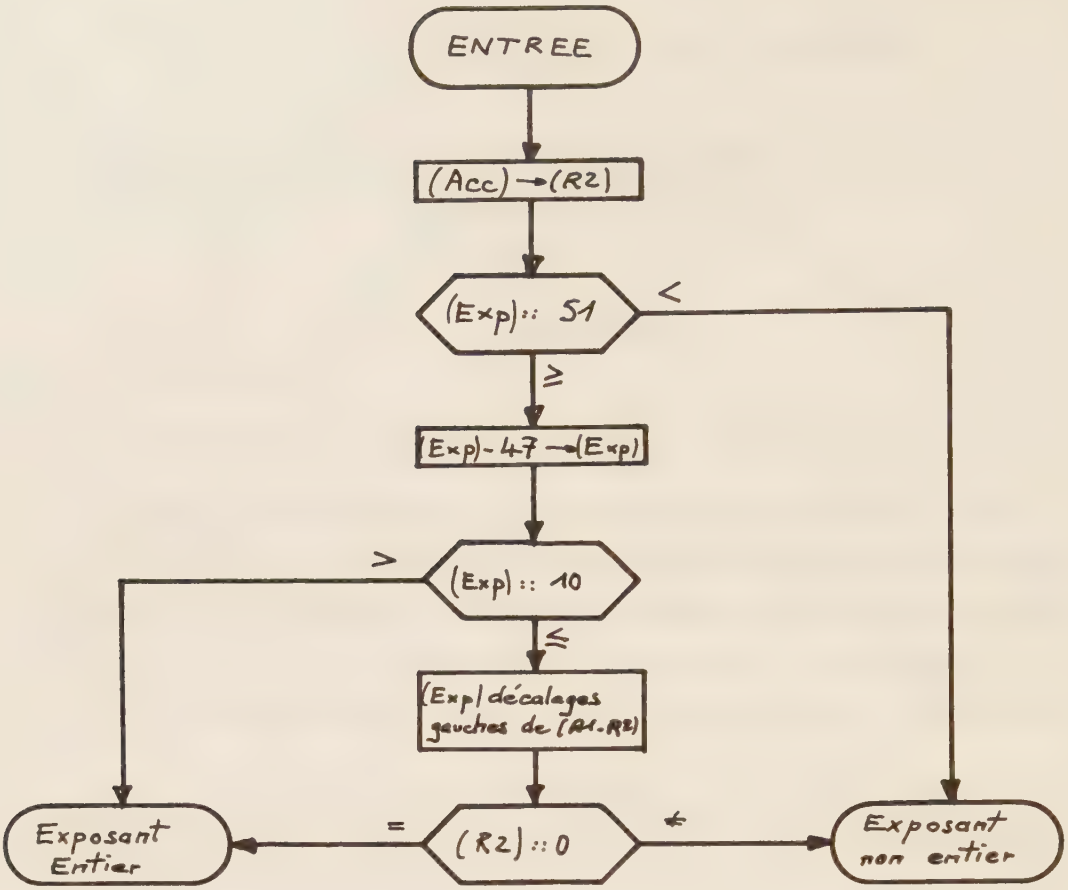


Figure E-3

E-3 - PARITE

Pour tester la parité, on se sert de la configuration des registres R1 et R2 (figure E-3), après qu'on ait trouvé que l'exposant était entier. Il suffit de décaler l'attelage R1-R2 d'une position vers la droite pour avoir en tête du registre R2 le chiffre de plus bas poids de la figure. L'intersection logique avec '10' permet de déterminer la parité ou l'imparité.

Exemples :

Forme externe	Forme interne	Après traitement 'Entier' R1	R2	Après décalage R1	R2	Après 'ET'	Conclusion
1	51 01 00 00 00	00 00 00 04 01	00 00 00 00 00	00 40	10,00	10,	Impair
-1	51 99 00 00 00	00 00 00 04 99	00 00 00 00 00	00 49	90,00	10,	Impair
4	51 04 00 00 00	00 00 00 04 04	00 00 00 00 00	00 40	40,00	00,	Pair
-4	51 96 00 00 00	00 00 00 04 96	00 00 00 00 00	00 49	60,00	00,	Pair
179	53 01 79 00 00	00 00 06 01 79	00 00 00 00 00	60 17	90,00	10,	Impair
180	53 01 80 00 00	00 00 06 01 80	00 00 00 00 00	60 18	00,00	00,	Pair
1234567	57 01 23 45 67	10 01 23 45 67	00 00 00 00 00	34 56	70,00	10,	Impair
12345670	58 01 23 45 67	00 00 00 00 00	11 01 23 45 67	00 00	01,10	00,	Pair

Figure E-4

Remarquons que dans le cas où Exp est supérieur à 10 (ce qui correspond à un exposant supérieur à 57 dans la forme interne), il n'y a pas eu de décalage, et on peut employer le même algorithme de recherche de parité que dans les autres cas, bien que l'on sache que le nombre est toujours pair (il y a des zéros à droite).

L'organigramme est alors le suivant :

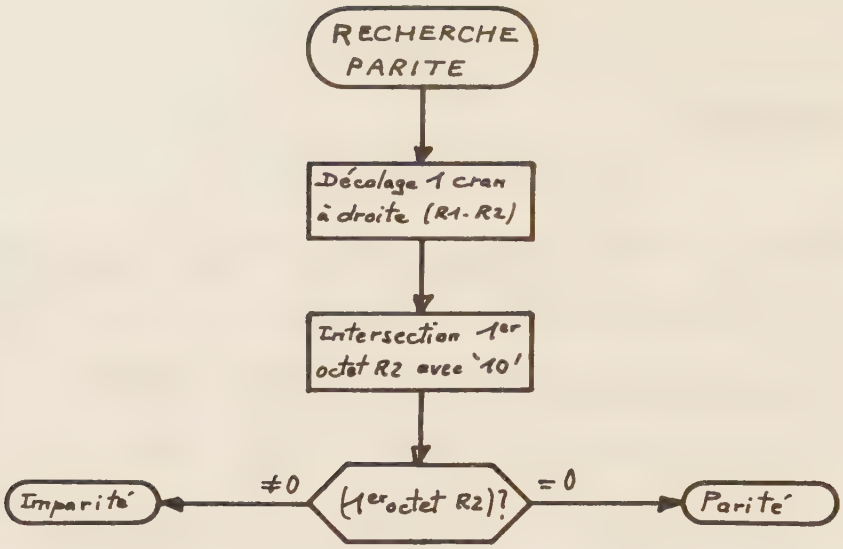

RECHERCHE PARITE
Décalage 1 cran à droite (R1-R2)
Intersection 1er octet R2 avec '10'
(1er octet R2)?
≠0
=0
Imparité
Parité

Figure E-5

ANNEXE F

SOUS-PROGRAMME DE CONVERSION EXTERNE EXTFLOCH

F-1 - BUT

De la même façon qu'il faut un sous-programme en entrée permettant de convertir des quantités numériques sous forme externe en représentation flottante interne, il faut un sous-programme capable de faire l'inverse en sortie.

Le sous-programme système EXTFLØ fourni par le constructeur a l'inconvénient de ne fournir qu'une seule forme externe, une quantité purement fractionnaire associée à une puissance de 10 :

$$S0,d_1d_2d_3d_4d_5d_6d_7Ese_1e_2$$

où S est le signe de l'ensemble
d_1 à d_7 les chiffres significatifs
s le signe de l'exposant
e_1 et e_2 les 2 chiffres de l'exposant

F-2 - TYPES DE FORMAT

On laisse au sous-programme EXTFLOCH le soin de choisir le format de sortie adéquat.

Les différents formats sont les suivants :

F-2-1 - Format entier : le nombre peut s'exprimer à l'aide de 1 à 7 chiffres.

F-2-2 - Format fractionnaire pur : le nombre s'exprime sous la forme $\pm 0.d_1d_2d_3d_4d_5d_6d_7$ où les zéros de queue sont supprimés.

F-2-3 - Format décimal sans exposant : le nombre s'exprime à l'aide de 7 chiffres sous la forme :

$$\pm d_1d_2d_3.d_4d_5d_6d_7$$

La position du point décimal est variable. Les zéros de queue sont éliminés.

F-2-4 - Format décimal avec exposant : on revient au format classique :

$$\pm d_1.d_2d_3d_4d_5d_6d_7E\pm e_1e_2$$

Le point décimal est fixe, les zéros de queue sont éliminés et la partie exposant reste toujours accollée au dernier chiffre significatif.

F-3 - TRAITEMENT

Il faut préciser ici un certain nombre de conventions nécessaires à la compréhension de l'organigramme. On utilise 2 registres de base A et B et 5 registres normaux R1 à R5.

A l'entrée dans le sous-programme, A et B sont quelconques, R3, R4 et R5 sont à zéro, R1 contient le nombre en représentation interne à convertir et R2, l'adresse de la zone de conversion dans laquelle on enregistre la représentation externe.

Il y a initialisation de la base A suivie de la mise à blanc de la zone de sortie (sur 15 caractères). Un exposant interne nul signifie par convention que le nombre est nul (passage par F511).

L'exposant interne étant non-nul, on recherche le signe de la figure en isolant par une intersection logique avec 'F0' le chiffre de signe de la représentation interne.

On isole la figure dans le registe R3 par décalage de 2 crans à gauche de l'attelage R2-R3, si le signe est moins (chiffre de signe égal à 9), on redresse en positif le registre R3 et on édite le signe '-' en sortie.

En F500, on retranche 50 à l'exposant interne isolé dans R2, on obtient alors l'exposant réel du nombre. Si cet exposant est nul, on a affaire à un nombre fractionnaire pur (A), il y a édition de '0.' suivie de celle de 7 chiffres et on supprime les zéros qui sont en queue (F510).

Si l'exposant est positif, et ≤ 7, on passe en F504 et on effectue autant de décalages à gauche qu'il y a d'unités dans l'exposant, plus un, pour tenir compte du chiffre de signe. La comparaison à zéro du registre R3 permet alors d'isoler le cas des nombres entiers de celui des nombres décimaux sans exposant.

Le nombre de chiffres d'un entier (B) est égal à la valeur de l'exposant, il suffit d'éditer ces chiffres en sortie.

Pour un nombre décimal sans exposant, le nombre de chiffres derrière le point décimal est égal à 7 moins la valeur de l'exposant. On supprime après édition des chiffres et du point décimal, les zéros de queue.

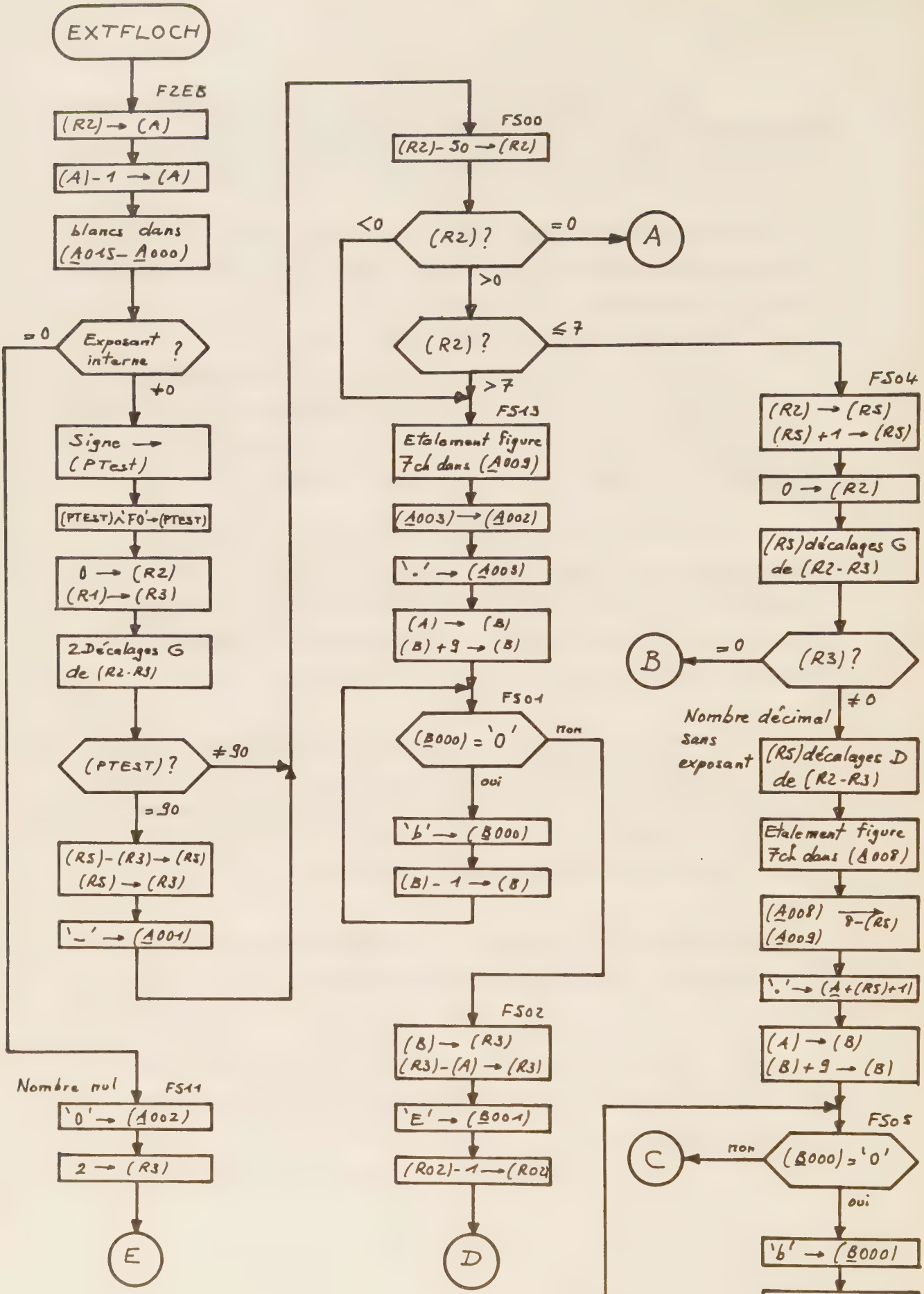

EXTFLOCH
FZEB
(R2) → (A)
(A) - 1 → (A)
blancs dans (A015 - A000)
Exposant interne ?
= 0
+0
Signe → (PTest)
(PTEST) ∧ F0' → (PTEST)
0 → (R2)
(R1) → (R3)
2 Décalages G de (R2-R3)
(PTEST) ?
≠ 90
= 90
(R5) - (R3) → (R5)
(R5) → (R3)
'_' → (A001)
Nombre nul
FS11
'0' → (A002)
2 → (R3)
E
FS00
(R2) - 30 → (R2)
(R2) ?
< 0
= 0
A
> 0
(R2) ?
≤ 7
> 7
FS13
Etalement figure 7ch dans (A009)
(A003) → (A002)
'.' → (A003)
(A) → (B)
(B) + 9 → (B)
FS01
(B000) = '0'
non
oui
'b' → (B000)
(B) - 1 → (B)
FS02
(B) → (R3)
(R3) - (A) → (R3)
'E' → (B001)
(R02) - 1 → (R02)
D
FS04
(R2) → (R5)
(R5) + 1 → (R5)
0 → (R2)
(R5) décalages G de (R2-R3)
B
= 0
(R3) ?
≠ 0
Nombre décimal sans exposant
(R5) décalages D de (R2-R3)
Etalement figure 7ch dans (A008)
(A008) (A009) 9 → (R5)
'.' → (A + (R5) + 1)
(A) → (B)
(B) + 9 → (B)
FS05
C
non
(B000) = '0'
oui
'b' → (B000)
(B) - 1 → (B)

Figure F-1 (a)

Enfin, pour les nombres avec exposant externe, on édite d'abord la figure avec mise en place du point décimal derrière le premier chiffre, puis l'exposant, en le décrémentant d'abord d'une unité (ce qui correspond au fait que l'on a $d_1.d_2..$ au lieu de $0.d_1d_2..$), puis en le redressant s'il est négatif avec édition du signe '-'.

Remarquons qu'au retour du sous-programme, le registre R3 contient le nombre de caractères édités, information qui est exploitée par le sous-programme d'impression pour calculer la dimension des champs dans la zone d'impression.

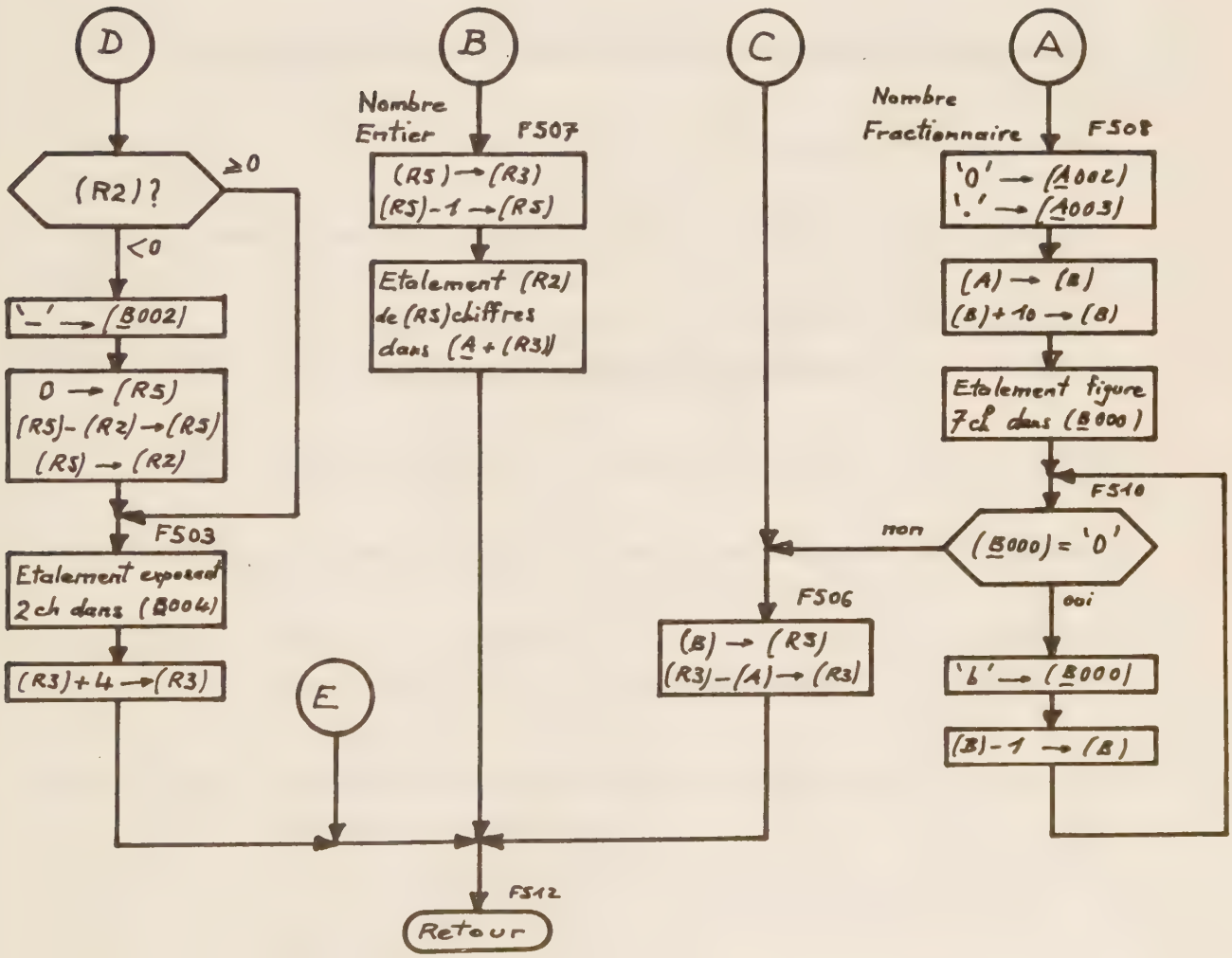


Figure F-1 (b)

ANNEXE G

LISTE DES ERREURS D'INTERPRETATION

Erreur n° 68 Encombrement [Programme + Matrices] trop grand .

Erreur n° 69 Tentative de calcul de A ↑ X avec A< 0, X ≠ entier .

Erreur n° 70 Variable simple non définie .

Erreur n° 71 Récursivité des fonctions FN∝ .

Erreur n° 72 Tentative de redéfinition de la fonction FN∝ .

Erreur n° 73 Fonction FN∝ non définie .

Erreur n° 74 Le nombre des fonctions FN∝ est trop grand (> 11) .

Erreur n° 75 Branchement sur ligne inexistante .
Recherche d'une ligne inexistante .

Erreur n° 76 Le numéro de ligne n'est pas valide .

Erreur n° 77 Erreur dans l'ordre RESTORE, il n'y a pas de DATA à la ligne indiquée .

Erreur n° 78 SORTIE DES DATA .

Erreur n° 79 Il y a une entrée illégale dans le corps d'une boucle .

Erreur n° 80 Le pas est nul dans la boucle .

Erreur n° 81 Le nombre de boucles emboîtées est trop grand (>10) .

Erreur n° 82 Tentative d'exécution d'un RETURN, sans qu'il y ait eu un GOSUB correspondant .

Erreur n° 83 Le nombre de sous-programmes encastrés est trop grand (> 20) .

Erreur n° 84 Oubli d'un indice .

Erreur n° 85 L'indice a une valeur plus grande que la dimension correspondante .

Erreur n° 86 Le redimensionnement impose un encombrement supérieur à l'ancien

Erreur n° 87 Il y a trop de tableaux (> 13) .

Erreur n° 88 La nouvelle dimension est négative ou nulle .

Erreur n° 89 Tentative de redéfinition d'un tableau .

Erreur n° 90 Tableau non défini .

Erreur n° 91 L'encombrement du tableau est trop grand (P x Q > 140) .

Erreur n° 92 Matrices de dimensions incompatibles dans multiplication d'une matrice par une expression .

Erreur n° 93 Matrices de dimensions incompatibles dans un ordre de recopie matricielle .

Erreur n° 94 Matrices de dimensions incompatibles dans un ordre d'addition matricielle .

Erreur n° 95 Matrice de dimensions incompatibles dans un ordre de produit matriciel .

Erreur n° 96 Matrice non carrée dans une mise à matrice unité .

Erreur n° 97 Matrices de dimensions incompatibles dans un ordre de transposition .

Erreur n° 98 Matrice 1 ou matrice 2 non-carrée, ou matrices de dimensions incompatibles dans un ordre d'inversion de matrice .

Erreur n° 99 Matrice singulière .

BIBLIOGRAPHIE

BULL-GENERAL ELECTRIC	BASIC Manuel de référence
CHEVANCE R.J	Compilation (notes de cours) - Institut de programmation
CONTROL-DATA	BASIC Language Reference Manual
DEBRAINE P.	Machines de Traitement de l'Information Tome III (à paraître)
EICKEL J, PAUL M, BAUER F.L, SAMELSON K.	A Syntax - Controlled generator of Formal Language Processors
FLOYD R.W	Syntactic Analysis and Operator Precedence Journal of ACM July 63
HEWLETT-PACKARD	A guide to time Shared BASIC
HOPGOOD F.R.A	Compiling Techniques - MacDonald - London
KEMENY (J.G) and KURTZ (T.E)	BASIC Programming John Wiley and Sons 1971
KUO	Numerical Methods and Computer Addison-Wesley 1965
NICOLAS M.	Notions sur les langages - Institut de Programmation
PELLETIER	Techniques numériques appliquées au calcul scientifique Masson 1971
RALSTON	Méthodes mathématiques pour calculateurs arithmétiques Dunod 1965
ROSSIENSKY M.	Théorie et Construction des Compilateurs - CEA - Rapports DCE Saclay n° 316 - 320 - 328

Vu et approuvé

Orsay, le 13 novembre 1972

Le Vice-Président du Centre d'Orsay